

版权注意事项：1、书籍版权归著者和出版社所有；  
2、本PDF仅用于个人获取知识，进行私底下知识交流；  
3、PDF获得者不得在互联网以任何目的进行传播；  
如有需要，请尽量购买正版实体书！支持书籍作者！！





# 网络爬虫全解析

## 技术、原理与实践

罗刚◎著



中国工信出版集团



电子工业出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>

## 作者简介

---

罗 刚 ▶

猎兔搜索创始人，带领猎兔搜索技术开发团队先后开发出猎兔中文分词系统、猎兔问答系统、猎兔信息提取系统、猎兔智能垂直搜索系统以及网络信息监测系统等，实现互联网信息的采集、过滤、搜索和实时监测。曾编写出版《自己动手写搜索引擎》、《自己动手写网络爬虫》、《使用C#开发搜索引擎》，获得广泛好评。在北京和上海等地均有猎兔培训的学员。

---

# 网络爬虫全解析

## 技术、原理与实践

罗刚◎著

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

## 内 容 简 介

本书介绍了如何开发网络爬虫。内容主要包括开发网络爬虫所需要的 Java 语法基础和网络爬虫的工作原理,如何使用开源组件 HttpClient 和爬虫框架 Crawler4j 抓取网页信息,以及针对抓取到的文本进行有效信息的提取。为了扩展抓取能力,本书介绍了实现分布式网络爬虫的关键技术。

另外,本书介绍了从图像和语音等多媒体格式文件中提取文本信息,以及如何使用大数据技术存储抓取到的信息。最后,以实战为例,介绍了如何抓取微信和微博,以及在电商、医药、金融等领域的案例应用。其中,电商领域的应用介绍了使用网络爬虫抓取商品信息入库到网上商店的数据库表。医药领域的案例介绍了抓取 PubMed 医药论文库。金融领域的案例介绍了抓取股票信息,以及从年报 PDF 文档中提取表格等。

本书适用于对开发信息采集软件感兴趣的自学者。也可以供有 Java 或程序设计基础的开发人员参考。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有,侵权必究。

## 图书在版编目(CIP)数据

网络爬虫全解析:技术、原理与实践/罗刚著. —北京:电子工业出版社,2017.3

ISBN 978-7-121-31071-3

I. ①网… II. ①罗… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆 CIP 数据核字(2017)第 047570 号

策划编辑:董 英

责任编辑:徐津平

印 刷:北京京师印务有限公司

装 订:北京京师印务有限公司

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编:100036

开 本:787×980 1/16 印张:27.75 字数:585 千字

版 次:2017 年 3 月第 1 版

印 次:2017 年 3 月第 1 次印刷

印 数:3000 册 定价:79.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888, 88258888。

质量投诉请发邮件至 zlts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式:(010) 51260888-819, faq@phei.com.cn。



# 前言

现代社会，有效信息对人来说就像氧气一样不可或缺。互联网让有效信息的收集工作变得更容易。当你在网上冲浪时，网络爬虫也在网络中穿梭，自动收集互联网上有用的信息。

自动收集和筛选信息的网络爬虫让有效信息的流动性增强，让我们更加高效地获取信息。随着越来越多的信息显现于网络，网络爬虫也越来越有用。

各行业都离不开对信息的采集和加工处理。例如，农业需要抓取气象数据、农产品行情数据等实现精准农业。机械行业需要抓取零件、图纸信息作为设计参考。医药行业需要抓取一些疾病的治疗方法信息。金融行业需要抓取上市公司基本面和技术面等相关信息作为股市涨跌的参考，例如，太钢生产出圆珠笔头，导致它的股票“太钢不锈”上涨。此外，金融行业也需要抓取股民对市场的参与度，作为市场大势判断的依据。

每个人都可以用网络爬虫技术获得更好的生存策略，避免一些糟糕的情况出现，让自己生活得更加幸福和快乐。例如，网络爬虫可以收集到二甲双胍等可能抗衰老的药物，从而让人生活得更加健康。

本书的很多内容来源于搜索引擎、自然语言处理、金融等领域的项目开发和教学实践。感谢开源软件的开发者们，他们无私的工作丰富了本书的内容。

本书从开发网络爬虫所需要的 Java 语法开始讲解，然后介绍基本的爬虫原理。通过介绍优先级队列、宽度优先搜索等内容，引领读者入门，之后根据当前风起云涌的云计算热潮，重点讲述了云计算的相关内容及其在爬虫中的应用，以及信息抽取、链接分析等内容。接下来介绍了有关爬虫的 Web 数据挖掘等内容。为了让读者更深入地了解爬虫的实际应用，最后一章是案

例分析。本书相关的代码在读者 QQ 群（294737705）的共享文件中可以找到。

本书适合需要具体实现网络爬虫的程序员使用，对于信息检索等相关领域的研究人员也有一定的参考价值，同时猎兔搜索技术团队已经开发出以本书为基础的专门培训课程和商业软件。目前的一些网络爬虫软件仍有很多功能有待完善，作者真诚地希望通过本书把读者带入网络爬虫开发的大门并认识更多的朋友。

感谢早期合著者、合作伙伴、员工、学员、家人的支持，他们给我们提供了良好的工作基础，这是一个持久可用的工作基础。在将来，希望我们的网络爬虫代码和技术能够像植物一样快速生长。

参与本书编写的还有崔智杰、石天盈、张继红、张进威、刘宇、何淑琴、任通通、高丹丹、徐友峰、孙宽，在此一并表示感谢。

罗 刚

2017 年 2 月

---

轻松注册成为博文视点社区用户（[www.broadview.com.cn](http://www.broadview.com.cn)），您即可享受以下服务：

- **下载资源：**本书所提供的示例代码及资源文件均可在【下载资源】处下载。
- **提交勘误：**您对书中内容的修改意见可在【提交勘误】处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **与作者交流：**在页面下方【读者评论】处留下您的疑问或观点，与作者和其他读者一同学习交流。

页面入口：<http://www.broadview.com.cn/31071>

二维码：



# 目 录

第 1 章 技术基础	1
1.1 第一个程序	1
1.2 准备开发环境	2
1.2.1 JDK	2
1.2.2 Eclipse	3
1.3 类和对象	4
1.4 常量	5
1.5 命名规范	6
1.6 基本语法	6
1.7 条件判断	7
1.8 循环	8
1.9 数组	9
1.10 位运算	11
1.11 枚举类型	13
1.12 比较器	14
1.13 方法	14
1.14 集合类	15
1.14.1 动态数组	15
1.14.2 散列表	15
1.15 文件	19

1.15.1 文本文件	19
1.15.2 二进制文件	23
1.16 多线程	27
1.16.1 基本的多线程	28
1.16.2 线程池	30
1.17 折半查找	31
1.18 处理图片	34
1.19 本章小结	35
第2章 网络爬虫入门	36
2.1 获取信息	36
2.1.1 提取链接	37
2.1.2 采集新闻	37
2.2 各种网络爬虫	38
2.2.1 信息采集器	40
2.2.2 广度优先遍历	41
2.2.3 分布式爬虫	42
2.3 爬虫相关协议	43
2.3.1 网站地图	44
2.3.2 Robots 协议	45
2.4 爬虫架构	48
2.4.1 基本架构	48
2.4.2 分布式爬虫架构	51
2.4.3 垂直爬虫架构	54
2.5 自己写网络爬虫	55
2.6 URL 地址查新	57
2.6.1 嵌入式数据库	58
2.6.2 布隆过滤器	60
2.6.3 实现布隆过滤器	61
2.7 部署爬虫	63
2.7.1 部署到 Windows	64
2.7.2 部署到 Linux	64
2.8 本章小结	65



第 3 章 定向采集 .....	69
3.1 下载网页的基本方法 .....	69
3.1.1 网卡 .....	70
3.1.2 下载网页 .....	70
3.2 HTTP 基础 .....	75
3.2.1 协议 .....	75
3.2.2 URI .....	77
3.2.3 DNS .....	84
3.3 使用 HttpClient 下载网页 .....	84
3.3.1 HttpCore .....	94
3.3.2 状态码 .....	98
3.3.3 创建 .....	99
3.3.4 模拟浏览器 .....	99
3.3.5 重试 .....	100
3.3.6 抓取压缩的网页 .....	102
3.3.7 HttpContext .....	104
3.3.8 下载中文网站 .....	105
3.3.9 抓取需要登录的网页 .....	106
3.3.10 代理 .....	111
3.3.11 DNS 缓存 .....	112
3.3.12 并行下载 .....	113
3.4 下载网络资源 .....	115
3.4.1 重定向 .....	115
3.4.2 解决套接字连接限制 .....	118
3.4.3 下载图片 .....	119
3.4.4 抓取视频 .....	122
3.4.5 抓取 FTP .....	122
3.4.6 网页更新 .....	122
3.4.7 抓取限制应对方法 .....	126
3.4.8 URL 地址提取 .....	131
3.4.9 解析 URL 地址 .....	134
3.4.10 归一化 .....	135

3.4.11 增量采集	135
3.4.12 iframe	136
3.4.13 抓取 JavaScript 动态页面	137
3.4.14 抓取即时信息	141
3.4.15 抓取暗网	141
3.5 PhantomJS	144
3.6 Selenium	145
3.7 信息过滤	146
3.7.1 匹配算法	147
3.7.2 分布式过滤	153
3.8 采集新闻	153
3.8.1 网页过滤器	154
3.8.2 列表页	159
3.8.3 用机器学习的方法抓取新闻	160
3.8.4 自动查找目录页	161
3.8.5 详细页	162
3.8.6 增量采集	164
3.8.7 处理图片	164
3.9 遍历信息	164
3.10 并行抓取	165
3.10.1 多线程爬虫	165
3.10.2 垂直搜索的多线程爬虫	168
3.10.3 异步 IO	172
3.11 分布式爬虫	176
3.11.1 JGroups	176
3.11.2 监控	179
3.12 增量抓取	180
3.13 管理界面	180
3.14 本章小结	181
第 4 章 数据存储	182
4.1 存储提取内容	182
4.1.1 SQLite	183

4.1.2 Access 数据库 .....	185
4.1.3 MySQL .....	186
4.1.4 写入维基 .....	187
4.2 HBase .....	187
4.3 Web 图 .....	189
4.4 本章小结 .....	193
第 5 章 信息提取 .....	194
5.1 从文本提取信息 .....	194
5.2 从 HTML 文件中提取文本 .....	195
5.2.1 字符集编码 .....	195
5.2.2 识别网页的编码 .....	198
5.2.3 网页编码转换为字符串编码 .....	201
5.2.4 使用正则表达式提取数据 .....	202
5.2.5 结构化信息提取 .....	206
5.2.6 表格 .....	209
5.2.7 网页的 DOM 结构 .....	210
5.2.8 使用 Jsoup 提取信息 .....	211
5.2.9 使用 XPath 提取信息 .....	217
5.2.10 HTMLUnit 提取数据 .....	219
5.2.11 网页结构相似度计算 .....	220
5.2.12 提取标题 .....	222
5.2.13 提取日期 .....	224
5.2.14 提取模板 .....	225
5.2.15 提取 RDF 信息 .....	227
5.2.16 网页解析器原理 .....	227
5.3 RSS .....	229
5.3.1 Jsoup 解析 RSS .....	230
5.3.2 ROME .....	231
5.3.3 抓取流程 .....	231
5.4 网页去噪 .....	233
5.4.1 NekoHTML .....	234



5.4.2	Jsoup	238
5.4.3	提取正文	240
5.5	从非 HTML 文件中提取文本	241
5.5.1	PDF 文件	242
5.5.2	Word 文件	245
5.5.3	Rtf 文件	247
5.5.4	Excel 文件	253
5.5.5	PowerPoint 文件	254
5.6	提取标题	254
5.6.1	提取标题的一般方法	255
5.6.2	从 PDF 文件中提取标题	259
5.6.3	从 Word 文件中提取标题	261
5.6.4	从 Rtf 文件中提取标题	261
5.6.5	从 Excel 文件中提取标题	267
5.6.6	从 PowerPoint 文件中提取标题	270
5.7	图像的 OCR 识别	270
5.7.1	读入图像	271
5.7.2	准备训练集	272
5.7.3	图像二值化	274
5.7.4	切分图像	279
5.7.5	SVM 分类	283
5.7.6	识别汉字	287
5.7.7	训练 OCR	289
5.7.8	检测行	290
5.7.9	识别验证码	291
5.7.10	JavaOCR	292
5.8	提取地域信息	292
5.8.1	IP 地址	293
5.8.2	手机	315
5.9	提取新闻	316
5.10	流媒体内容提取	317
5.10.1	音频流内容提取	317

5.10.2 视频流内容提取 .....	321
5.11 内容纠错 .....	322
5.11.1 模糊匹配问题 .....	325
5.11.2 英文拼写检查 .....	331
5.11.3 中文拼写检查 .....	333
5.12 术语 .....	336
5.13 本章小结 .....	336
第 6 章 Crawler4j .....	338
6.1 使用 Crawler4j .....	338
6.1.1 大众点评 .....	339
6.1.2 日志 .....	342
6.2 crawler4j 原理 .....	342
6.2.1 代码分析 .....	343
6.2.2 使用 Berkeley DB .....	344
6.2.3 缩短 URL 地址 .....	347
6.2.4 网页编码 .....	349
6.2.5 并发 .....	349
6.3 本章小结 .....	352
第 7 章 网页排重 .....	353
7.1 语义指纹 .....	354
7.2 SimHash .....	357
7.3 分布式文档排重 .....	367
7.4 本章小结 .....	369
第 8 章 网页分类 .....	370
8.1 关键词加权法 .....	371
8.2 机器学习的分类方法 .....	378
8.2.1 特征提取 .....	380
8.2.2 朴素贝叶斯 .....	384
8.2.3 支持向量机 .....	393
8.2.4 多级分类 .....	401

8.2.5 网页分类	403
8.3 本章小结	403
第9章 案例分析	404
9.1 金融爬虫	404
9.1.1 中国能源政策数据	404
9.1.2 世界原油现货交易和期货交易数据	405
9.1.3 股票数据	405
9.1.4 从 PDF 文件中提取表格	408
9.2 商品搜索	408
9.2.1 遍历商品	410
9.2.2 使用 HttpClient	415
9.2.3 提取价格	416
9.2.4 水印	419
9.2.5 数据导入 ECShop	420
9.2.6 采集淘宝	423
9.3 自动化行业采集	424
9.4 社会化信息采集	424
9.5 微博爬虫	424
9.6 微信爬虫	426
9.7 海关数据	426
9.8 医药数据	427
9.9 本章小结	429
后记	430

# 1

## 第 1 章

### 技术基础

很多种编程语言都可以用来开发爬虫。相对于 Python，Java 由于严谨的语法结构和体系结构，所以在开发爬虫方面有后发优势。

很多网络爬虫是使用 Java 或者 C#语言开发的。如果是开发采集器那样的客户端爬虫，那么可以使用 C#开发爬虫。如果是运行在服务器端的爬虫，则可以用 Java 开发。

只要有目标，你可以做到很多从来没有做过的事情。没有基础也可以学习开发网络爬虫，本章是专门为开发爬虫写的 Java 基础介绍。

#### 1.1 第一个程序

Java 程序都运行在虚拟机上。为什么要用虚拟机，而不是直接运行在本机的操作系统上？因为 Windows 是收费的，而 Linux 可以免费使用。可以把 Windows 当作开发环境使用，而把程序部署在 Linux 上。因为运行在指令集相同的虚拟机上，所以 Java 程序可以不经修改地在不同



操作系统之间切换。

并不一定要自己买房子以后才有地方住。并不一定要在本机安装开发环境以后，才能运行第一个 Java 程序。有一些在线的开发环境可运行 Java 程序，例如 <http://ideone.com/>。

第一个 Java 程序是从一个类中定义的 `main` 方法开始执行的。

```
public class Crawler{  
    public static void main (String args[]) {  
        System.out.println("Hello Crawler!");  
    }  
}
```

底层到底做了些什么？源代码定义了一个叫作 `Crawler` 的类，虚拟机执行其中的 `main` 方法。

## 1.2 准备开发环境

Eclipse 也是使用 Java 开发的，所以先准备基本的 Java 开发环境（简称 JDK），然后准备运行在 JDK 上的 Eclipse。

### 1.2.1 JDK

JDK 可以从 Java 官方网站 <http://java.sun.com> 下载得到。注意，不是从 <http://www.java.com> 下的 Java 虚拟机。

下载 Java SE，也就是标准版本。Latest Release 是最新发布的安装程序。因为可以在 Windows 或 Linux 等多种操作系统环境下开发 Java 程序，所以有多个操作系统的 JDK 版本供选择。

因为 JDK 是有版权的，所以需要接受许可协议（Accept License Agreement）后才能下载。下载完毕后，使用默认方式安装 JDK 即可。JDK 相关的文件都放在一个叫作 `JAVA_HOME` 的根目录下。JDK 根目录的命名格式是 `C:\Program Files\Java\jdk1.6.0_<version>`，最后以一个数字类型的版本号结尾，例如 10 或者 21。

因为一台机器可以安装多个 JDK 和 JVM，为了避免混乱，可以新增环境变量 `JAVA_HOME`，指定一个默认使用的 JDK。



使用 `echo` 命令检查环境变量 `JAVA_HOME`。

```
>echo %JAVA_HOME%  
C:\Program Files\Java\jdk1.6.0_10
```

如果只需要使用集成开发环境，配置 `JAVA_HOME` 环境变量就可以了。为了检查 `JAVA_HOME` 是否已经正确设置，在任何路径输入 Java 命令 “`>java -version`” 显示虚拟机的版本号就可以了。

```
java version "1.6.0_10-rc"  
Java(TM) SE Runtime Environment (build 1.6.0_10-rc-b28)  
Java HotSpot(TM) Client VM (build 11.0-b15, mixed mode, sharing)
```

如果还需要在控制台下执行，则需要访问编译程序的 `javac.exe` 或者执行 Java 类的 `java.exe`。环境变量 `PATH` 指定了从哪里找 `java.exe` 这样的可执行文件。可以从多个路径查找可执行文件，这些路径以分号隔开。如果想在命令行运行 Java 程序，还可以修改已有的环境变量 `PATH`，增加 Java 程序所在的路径，例如 `C:\Program Files\Java\jdk1.6.0_10\bin`。

然后检查环境变量 `PATH`。

```
>echo %PATH%
```

为了检查 `PATH` 是否已经正确设置，在任何路径输入 `javac` 命令显示 `javac` 的用法就可以了，也可以用第一个 Java 程序试验下。

```
>javac Crawler.java  
>java Crawler
```

运行看是否显示 “Hello Crawler!”。

## 1.2.2 Eclipse

就好像理发有推子等专门的理发用具，开发软件也有专门的集成开发环境。开发 Java 程序最流行的工具叫作 Eclipse (<http://www.eclipse.org>)。

Eclipse 也有很多版本，可以选择最简单的一个版本——Eclipse IDE for Java Developers。Eclipse 是绿色软件，无须安装，解压后就可以直接使用。在 Windows 下，双击就可以解压文件。如果需要专门的解压软件，推荐使用 7z (<http://www.7-zip.org/>)。

Eclipse 默认是英文界面，如果习惯用中文界面，可以从 <http://www.eclipse.org/babel/>

downloads.php 上下载支持中文的语言包。

Eclipse 把软件按项目管理，每个项目都有自己的.classpath 文件，指定了源代码路径、编译后输出文件的路径以及这个项目引用的 jar 包的路径。

## 1.3 类和对象

世界上有各种各样的生物，每个生物都属于某一个物种。蜘蛛是一个物种，每个蜘蛛都是一个对象。同样，Java 虚拟机的内存中也有很多各种各样的对象。

使用类可以创建具有相同结构和行为的对象。打印 Hello World 的例子并没有创建对象，因为 main 是一个静态方法，不属于任何一个类。现在创建一个属于对象的方法。

```
public class Spider{
    public void hello(){
        System.out.println("Hello World!");
    }
    public static void main (String args[]) {
        Spider p = new Spider(); //新建一个对象
        p.hello(); //调用 hello 方法
    }
}
```

Java 源文件的扩展名为.java，而且必须与类名相同。上面这个 Spider 类必须放在 Spider.java 文件中。

每个人都由不同的原子构成。每个对象都占据不同的内存空间。使用关键字 new 来为对象分配空间，就是实例化对象。关键字 new 声明了对象的诞生，但是不是所有的数据类型都是对象。一些基本的数据类型，例如 int、boolean 等都不是对象，不能用 new 的方式实例化。

toString 方法返回一个描述对象内部状态的字符串。所有的对象都有 toString 方法，这些共同的方法在 Object 类中定义，Object 是所有对象的共同祖先。

有的对象专门用来存储数据，叫作 POJO 类，还有些用来执行任务，例如，爬虫类或者搜索类。

在接口中只能定义一些方法和常量。定义一个接口 `Visitor` 处理碰到的网址：

```
String seed = " http://wallstreetcn.com/"; //抓取种子：华尔街新闻网

void expand(URL seed, visitor){
}
```

`Visitor` 的具体实现类处理抓下来的网页内容。

## 1.4 常量

一般把常量值声明成 `final` 类型，表示以后不会再修改它。如果需要给这个常量动态赋值，则可以放在 `static{}` 程序块中。而且最好放在一个单独的类中，这样其他的类就可以把它当成静态常量访问。

```
public final class Constants {
    public static final String OS_ARCH = System.getProperty("os.arch");

    public static final boolean JRE_IS_64BIT; //常量
    static {
        String x = System.getProperty("sun.arch.data.model");
        if (x != null) {
            JRE_IS_64BIT = x.indexOf("64") != -1;
        } else {
            if (OS_ARCH != null && OS_ARCH.indexOf("64") != -1) {
                JRE_IS_64BIT = true;
            } else {
                JRE_IS_64BIT = false;
            }
        }
    }
}
```



## 1.5 命名规范

类的首字母大写，例如，新闻爬虫叫作 NewsSpider，论坛爬虫叫作 ForumSpider。

最好用英语命名，不要用拼音，因为拼音容易有歧义。Libai 可能表示李白，也可能表示立白。为了和国际接轨，程序最好能让讲英语的程序员看懂。Java 中的关键词都是英文，既然不能把 if 写成 ruguo（如果），所以类名、变量名、方法名也应该用英语命名。

可以使用 Google 机器翻译 (<http://translate.google.cn>)。虽然机器翻译可能把“公共卫生间”翻译成“Between public health”，但是大部分词的翻译结果还是比较靠谱的。

有很多种不同的名称，例如类的名称、变量的名称，它们的命名方式各不一样。爬虫类名 Crawler，以大写字母开头，其中的方法名为 getURLs，以小写字母开头。总的来说，有两种命名方式：以小写字母开头的命名方式和以大写字母开头的命名方式。

一个名称由多个单词组成，因为单词之间不允许有空格，所以用大小写的方式来区分单词间隔，如果都是大写字母，则单词之间用下划线隔开。

类名的单词首字母要大写，而常量名所有字母都要大写。

```
public static final double PI=3.14;    //圆周率
public static final double NEGATIVE_INFINITY = -1.0 / 0.0; //最大的负数
```

仅处理网址的类叫作 Handler。从网址提取内容的类叫作 Extractor。后续的代码实现会遵循一致的命名规范。

## 1.6 基本语法

从语法上讲，Java 语言和 C 语言非常相似，只是在细节上有一些差别。实际上，C 语言和 Java 语言的主要差别不在语言本身，而是在它们所执行的平台上。

Java 程序需要 JRE（Java Runtime Environment）运行环境来执行代码，但是 JRE 只限于在

Java 这一门语言中使用。

Java 源代码可以被编译成字节代码的一种中间状态，然后由已提供的虚拟机来执行这些字节代码。

和 C 一样，Java 的每一个应用程序都应该有一个入口点，表明该程序从哪里开始执行。为了让系统能找到入口点，入口方法名规定为 `main`。

```
class HelloWorld{  
    public static void main() {  
        System.out.println ("Hello World");  
    }  
}
```

Java 的每个类都可以有一个 `main` 方法，因此可以执行，但是往往需要有很多可以直接执行的测试代码。可以把这些功能不同的代码封装在不同的方法中，这样至少避免了大量的注释代码。

有些方法处理的方式相同，只是输入参数不一样，可以把这些方法命名成同一个名称，这叫作方法重载。例如，分词类可以切分一个字符串，或者一个文件。

```
Public class Segmenter{  
    Public Split(String sentence){}  
    Public Split(File file){  
        //内部调用 Split(String)  
    }  
}
```

## 1.7 条件判断

判断一个网址是否是详细页，如果是详细页，就从这个网址提取正文。

```
if(isDetail(url)){ //判断是否是详细页  
    extractContent(url); //提取正文  
}
```

## 1.8 循环

for 循环总是可以写成等价的 while 循环。

<pre>for (init-stmt; condition; next-stmt) {     body }</pre>	<pre>init-stmt; while (condition) {     body     next-stmt; }</pre>
---	---

for 语句中有三个子句。

- 在循环开始之前执行 `init-stmt` 语句，通常在这里初始化迭代变量。
- 在每次循环之前，测试 `condition` 表达式。如果布尔表达式是 `false`，就不会执行循环（和 `while` 循环一样）。
- 在 `body` 执行后，执行 `next-stmt` 语句。一般在这里增加迭代变量的值。

例如，使用 for 循环生成出要遍历的网址。

```
for(int pageNum = 2;pageNum<20;++pageNum){  
    String pattern =  
        "http://roll.finance.sina.com.cn/finance/lcl/cfgs/index_%d.shtml";  
    String url = String.format(pattern, pageNum );  
    System.out.println(url);  
}
```

for 循环的一种特殊写法：

```
for(;;){  
    //代码  
}
```

把 for 循环当成一个永远为真的循环来使用时，它等价于：

```
while(true){  
    //代码  
}
```



for-each 循环, 看起来简洁, 但是中间的元素一般是如何定义的不是那么容易理解, 如下所示。

```
for (double[] pArray : prob)
for (double p : pArray)
System.out.println(p);
```

元素的名字随便起, 根据 iterator 接口返回值的类型定义, 要实现 iterator 接口的对象才能够用 for-each 循环遍历。

## 1.9 数组

军训中的 8×8 方阵可以表示成二维数组。

```
Person[][] matrix = new Person[8][8];
```

matrix 是二维数组的名字, 而从 matrix[0]到 matrix[7]都是一维数组, matrix[0]包含 8 个对象。所以可以把二维数组看成由一维数组组成的一维数组。

使用 new 关键字创建数组时, 可以使用变量声明数组的长度。

```
int num = 8;
Person[] personArray = new Person[num];
```

对于使用变量长度创建数组的数组, 在创建数组的时候需要知道数组的长度。水浒故事的原型是以宋江为首的 36 人, 到后来的小说中发展成为 108 人。如果用一个数组表示其中的每个人, 在创建时仍然不知道后来需要存储多少元素, 所以需要真正意义上的动态数组。实现长度可变的动态数组时需要把数据从长度小的数组复制到新的更长的数组上。

复制数组实现起来并不困难, 但是如果数组长度很大, 则需要考虑移动的性能。高速铁路有专门的客运专线, 能把大量的人从一个地方快速运到另外一个地方。System.arraycopy()方法专门用来快速移动数组中的数据, 它把指定个数的元素从源数组复制到目的数组。源数组和目的数组中的元素类型必须相同。

这个方法有 5 个参数。

- src: 源数组。

- srcPos: 源数组要复制的起始位置。
- dest: 目的数组。
- destPos: 目的数组放置的起始位置。
- length: 复制的长度。

在动态数组中使用 `System.arraycopy` 方法。

```
public class DynamicArrayOfInt {
    private int[] data; //保存数据的数组

    public DynamicArrayOfInt() { //构造器
        data = new int[1]; //按需增长的数组
    }

    public int get(int position) {
        //得到数组中指定位置的值
        return data[position];
    }

    public void put(int position, int value) {
        //把值存储到数组中指定位置
        //为了包含这个位置, 如果需要, 数据数组大小会增长
        if (position >= data.length) {
            //如果指定的位置超出了数据数组的实际大小, 则把数组的大小翻倍
            //如果仍然不包含指定的位置, 则新的大小设置成 2*position
            int newSize = 2 * data.length;
            if (position >= newSize)
                newSize = 2 * position;
            int[] newData = new int[newSize];
            System.arraycopy(data, 0, newData, 0, data.length); //复制内容到新的数组
            data = newData; //用新数组代替原来的数组
        }
        data[position] = value;
    }
}
```

例如, 使用单个元素赋值的方法复制一个二维数组。

```
static int[][] copy2D(int[][] in){
    int[][] ret = new int[in.length][in[0].length];
    for(int i = 0; i < in.length; i++) {
```



```

    for(int j = 0;j < in[0].length;j++) {
        ret[i][j] = in[i][j];
    }
}
return ret;
}

```

使用 `System.arraycopy` 方法快速复制二维数组。

```

static int[][] fastCopy2D(int[][] in){
    int[][] ret = new int[in.length][in[0].length];
    for(int i = 0;i < in.length;i++) {
        System.arraycopy( in[i], 0, ret[i], 0, in[0].length );
    }
    return ret;
}

```

如果把一个二维数组作为 `System.arraycopy` 的参数，则会把这个二维数组看成一维数组，只不过其中的每个元素都是一个一维数组。所以下面这样的写法不是深度复制，只是复制一维数组的引用。

```

static int[][] fastCopy2D(int[][] in){
    int[][] ret = new int[in.length][in[0].length];
    System.arraycopy( in, 0, ret, 0, in.length ); //不是深度复制
    return ret;
}

```

有些动态规划算法把计算的中间结果存储在二维数组中。例如，创建一个存储概率的二维数组。

```

int stageLength = 10; //第一维的长度
int types = 20;
double[][] prob = new double[stageLength][types]; //二维数组

```

## 1.10 位运算

计算机只认识 0 和 1 序列，也就是二进制。例如，农历七月七日的七夕，写成 7.7，转化成二进制就是 111.111。

二进制数组是最底层的数据结构。很多快速计算都会用到它，可以在二进制数组上进行位运算。

32 位或者 64 位的二进制数组可以用一个 `int` 或者 `long` 类型的数表示。很长的二进制位数组可以用 `BitSet` 类型的数表示。

作为常量的二进制数组写起来会很长，所以经常用到十六进制表示常量，十六进制以 `0x` 开头，如 `0xff` 就是 `11111111` 的十六进制表示，注意，这里是零 X，不是 OX。

位的逻辑运算有与、或、非、异或四种。例如，与运算取得低 8 位值。

```
uint low = (uint)(j) & 0xff; //取得变量 j 的低 8 位值
```

通过位移取得中间 8 位的值。

```
((uint)(j) >> 8) & 0xff;
```

异或运算取得二进制数组有差别的位。

```
long x = 0xfe07;
long y = 0x07;
long val = x ^ y; //异或运算
System.out.println(Long.toBinaryString(val)); //输出 1111111000000000
```

海明距离 (HammingDistance) 是针对长度相同的字符串或二进制数组而言的。对于二进制数组 `s` 和 `t`，`H(s, t)` 是两个数组对应位有差别的数量。例如，`1011101` 和 `1001001` 的海明距离是 2。

可以把两个无符号整型数按位异或 (XOR)，统计 `val` 中 1 的个数，结果就是海明距离，如表 1-1 所示。

表 1-1 计算海明距离

数 字	二进制表示
2	00000010
5	00000101
<XOR Result>	00000111

计算两个数的海明距离。

```
public static int hammingDistance(int x, int y){
    int dist = 0; //海明距离
```

```

int val = x ^ y; //异或结果
//统计val中1的个数
while (val>0) {
    ++dist;
    val &= val - 1; //去掉val中最右边的一个1
}
return dist;
}

```

## 1.11 枚举类型

为了比输赢，我们经常使用猜拳游戏：石头、剪刀、布，每次只能出其中的一种。当一个对象的取值范围是固定的一些值，往往使用枚举类型。

```

public enum GameValue{
    stone,      //石头
    scissors,   //剪刀
    cloth       //布
}

```

词有名词或动词等类别，句子有陈述句或疑问句等类型。使用枚举比使用无格式的整数来描述这些类型至少有如下三个优势。

- 枚举可以使代码更易于维护，有助于确保给变量指定合法的、期望的值。
- 枚举使代码更清晰，允许用描述性的名称表示整数值，而不是用含义模糊的数来表示。
- 枚举使代码更易于键入。在给枚举类型的实例赋值时，集成开发环境 Eclipse 会通过智能感知功能弹出一个包含可接受值的列表框，减少了按键次数，并能够让我们回忆起可能的值。

我们可以把词的类型或句子的类型定义成枚举类型。

```

public enum PartOfSpeech{ //词类别
    a, //形容词
    n, //名词
    v, //动词
}

```



## 1.12 比较器

字符串可以比较大小，方法就是从前往后逐个比较字符的编码值，但是不能用操作符“>”或者“<”比较大小。调用字符串对象的 `compareTo` 方法，代码如下所示。

```
String word = "yelp";
String anotherWord = "learn";
word.compareTo(anotherWord);
```

因为 `String` 类实现了 `Comparable` 接口，所以可以调用字符串对象的 `compareTo` 方法。`Comparable` 接口只定义了一个 `compareTo` 方法。

```
public interface Comparable<T> {
    public int compareTo(T o); //比较当前对象 this 和传入对象
}
```

`compareTo` 方法返回一个整数值，用来表示当前对象和传入的对象比较大小的结果。如果返回正数，则表示当前对象大于传入对象。如果返回负数，则表示当前对象小于传入对象。如果返回 0，则表示当前对象和传入对象相等。

## 1.13 方法

网页源代码中的 URL 可能是相对于当前网址本身的相对地址。定义一个叫作 `resolveUrl` 的方法用于把相对路径转为绝对路径。

```
public static String resolveUrl(final String baseUrl, final String relativeUrl) {
}
```

为了减轻记忆负担，同样功能的方法最好用相同的名字命名，哪怕需要传入的参数形式不一样。例如 `resolveUrl` 的另外一种实现。

```
private static Url resolveUrl(final Url baseUrl, final String relativeUrl) {
    //完成功能
}
```

```
public static String resolveUrl(final String baseUrl, final String relativeUrl) {  
    //调用 resolveUrl(url,string)完成功能  
}
```

如果相对 URL 以 `http://` 开头, 则直接返回这个 URL。实现 `resolveUrl` 方法基本上可以看成由一个字符串得到另外一个字符串。可以用有限状态转换的方法得到完整的 URL 地址。

## 1.14 集合类

把遍历过的网址放入一个集合:

```
HashSet<String> urlSeen = new HashSet<String>();  
urlSeen.put("http://www.lietu.com");
```

文档是单词的集合。搜索结果集也是文档的集合。所以在搜索引擎开发中, 集合类必不可少。集合类有动态数组 `ArrayList`、队列 `Queue`、堆栈 `Stack`, 以及存储键/值对的 `HashMap`。`HashMap` 是散列表的实现。

### 1.14.1 动态数组

把所有的网址放入动态数组中:

```
ArrayList<String> urls = new ArrayList<String>();  
urls.add("http://www.lietu.com");  
  
List<String> ipList = new ArrayList<String>();
```

用到的是泛型吗? 是的。

`List` 是个抽象的接口。`ArrayList` 是具体的实现类。因为是列表, 所以不确定数据的类型。

### 1.14.2 散列表

爬虫发送给 Web 服务器的网页请求中有若干项说明。`User-Agent` 和对应的值叫作键/值对。

把键/值对存储在 **HashMap** 中，就可以通过英文单词找到中文单词了。

```
//创建一个存储键/值对的散列表
HashMap<String,String> heads = new HashMap<String,String>();
heads.put("User-Agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"); //放入一个键/值对
```

通过 **User-Agent** 找到对应的值，也就是通过 **HashMap.get** 方法取得键对应的值。

```
System.out.println(heads.get("User-Agent")); //输出: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

如果要找的键不在 **HashMap** 中，则 **get** 方法返回空。

```
System.out.println(heads.get("hello")); //输出: null
```

遍历 **HashMap**。

```
//用 foreach 循环遍历元素
for (Entry<String, Integer> e : heads.entrySet()) {
    System.out.println(e.getKey() + "->" + e.getValue());
}
```

如果把自定义对象作为键对象，需要重写 **hashCode** 和 **equals** 方法。**HttpHost** 类重写了 **hashCode** 和 **equals** 方法。

```
public final class HttpHost{
    /** The default scheme is "http". */
    public static final String DEFAULT_SCHEME_NAME = "http";

    /** The host to use. */
    protected final String hostname;

    /** The lowercase host, for {@link #equals} and {@link #hashCode}. */
    protected final String lcHostname;

    /** The port to use. */
    protected final int port;

    /** The scheme (lowercased) */
    protected final String schemeName;

    public boolean equals(final Object obj) {
```



```

    if (this == obj) return true;
    if (obj instanceof HttpHost) {
        HttpHost that = (HttpHost) obj;
        return this.lcHostname.equals(that.lcHostname)
            && this.port == that.port
            && this.schemeName.equals(that.schemeName);
    } else {
        return false;
    }
}

/**
 * @see java.lang.Object#hashCode()
 */
public int hashCode() {
    int hash = LangUtils.HASH_SEED;
    hash = LangUtils.hashCode(hash, this.lcHostname);
    hash = LangUtils.hashCode(hash, this.port);
    hash = LangUtils.hashCode(hash, this.schemeName);
    return hash;
}
}

```

使用 `HttpHost` 对象作为键，对应的评分作为值。

```

HashMap<HttpHost, Integer> rankMap = new HashMap<HttpHost, Integer>();
rankMap.put(new HttpHost("lietu.com", 8);

```

`HashSet` 用来存储一个元素的集合。从名称可以看出，它是基于散列表的。但是只存储键，而不存储值。`HashSet` 中的 `Set` 是数学意义上的集合，而 `Java` 中的集合类则是一种更广义的称呼。

`HashSet` 其实使用 `HashMap` 来实现。为了判断某个键是否存在，它定义了一个叫作 `PRESENT` 的特殊对象。

```

static final Object PRESENT = new Object();

```

所有的键统一都对应 `PRESENT` 这一个对象。

`HashMap` 是散列表的实现。它存储了一个元素的集合。文档是单词的集合，搜索结果集也是文档的集合，所以在搜索引擎开发中，集合类必不可少。集合类除了 `HashMap`，还有动态数组 `ArrayList`、队列 `Queue` 和堆栈 `Stack` 等。增加一个元素到集合类调用 `add` 方法，增加键/值对

调用 put 方法。

存放在集合类里的元素都必须是对象。但是一些基本的数据类型如 int、boolean 等都不是对象。为了存放这些基本的数据类型，需要把它们封装成类，比如 int 封装成 Integer 类，boolean 封装成 Boolean 类。定义这些对象就是为了能够把基本数据类型当作对象来使用。Integer 对象比较值是否相等也是调用 equals 方法。

把基本类型用相应的引用类型包装起来，使其具有对象的性质。例如，int 包装成 Integer、float 包装成 Float，包装这个步骤叫作装箱。可以对 Character 类型的变量直接赋 char 类型的值，不用加强制类型转换。装箱约定的例子如下所示。

```
Character c = 'a';    //char 类型可以自动装箱成 Character 类型
Integer a = 100;     //这也是自动装箱
```

编译器调用 Integer.valueOf(int i)方法实现自动装箱。

和装箱相反，将引用类型的对象简化成值类型的数据叫作拆箱。

```
int b = new Integer(100); //这是自动拆箱
```

对于 Double 类型，可以调用 doubleValue 方法拆箱。

```
Double d = new Double(0);    //装箱
double x = d.doubleValue();  //拆箱
```

要知道集合中有多少个元素，可以使用 Collection 中定义的 size 方法。所有集合类都实现了这个接口。动态数组 ArrayList 是最常见的集合类，可以通过 ArrayList.size() 知道数组的长度。用 ArrayList.clear() 方法清空其中的元素，但是数组仍在，这样可以避免重复分配内存。集合类之间的关系如图 1-1 所示。

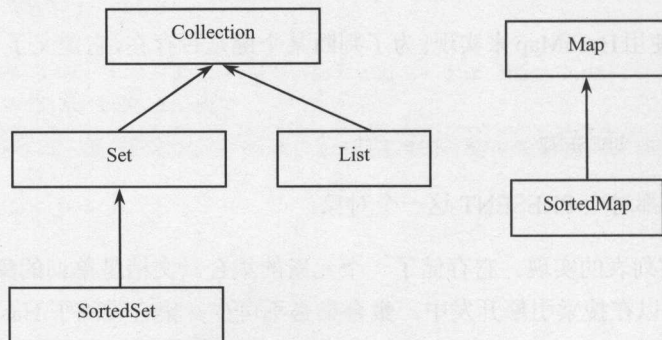


图 1-1 集合类之间的关系



## 1.15 文件

一本电子书往往就是操作系统中的一个文件。文件都是二进制格式的。但是也可以专门存储字符串，这样的文件叫文本文件。例如，网页往往以文本文件的形式存放在 Web 服务器中。文本文件可以直接用记事本编辑。大的文本文件如果用记事本打开需要很长时间，所以最好用写字板打开超过几兆以上的文件。可以用 UltraEdit 打开二进制格式的文件。

一般使用串行方式读出或者写入文件。总的来说，使用输入流把文件内容读入内存，使用输出流把内存中的信息写出到文件。这些类位于 java.io 包下。输入和输出的类和方法往往是对应的，例如，Reader 和 Writer 类对应。

Windows 系统文件大小经常以字节为单位。文件大小往往用 MB 或者 GB 衡量。1K 表示 1024，而 1M 表示 1024K，1G 表示 1024M。大约的计算方法是：1K 是 3 个零，1M 是 6 个零，1G 是 9 个零。

### 1.15.1 文本文件

先了解如何读写文本文件，然后看如何读写二进制文件。java.io.Reader 用来读取字符，它的子类 FileReader 用来读取文本文件。

FileReader 打开指定路径下的文件。文件的路径分隔符可以用 “\” 或者 “/” 表示。“\” 是 Windows 风格的写法，因为字符串中的特殊字符要转义，所以用两个斜线表示一个斜线。

```
FileReader fr = new FileReader("c:\\autoexec.bat"); //打开文本文件
```

“/” 是 Linux 风格的路径写法，因为不需要转义，所以正斜线只需要写一个就可以了。

```
FileReader fr = new FileReader("c:/autoexec.bat"); //打开文本文件
```

Linux 风格的和 Windows 风格的写法是等价的。

如果有一堆砖要搬，一次取不完，不会一次只拿一块砖，会尽量多拿几块。如果有很多内容要读，不会一次只读一个字节，而是一次尽量多读一些字节到缓存。

```
FileReader fr = new FileReader("c:/autoexec.bat"); //打开文本文件
```

```
BufferedReader br = new BufferedReader(fr); //缓存读
String line;
while((line = br.readLine()) != null) { //按行读入文件
    System.out.println(line);
}
fr.close(); //关闭文本文件
```

输入流把数据从硬盘读入随机访问存储器（Random Access Memory，简称 RAM）。可以根据输入流构建 **BufferedReader**，实现代码如下所示。

```
String fileName = "SDIC.txt"; //文件名
InputStream file = new FileInputStream(new File(fileName)); //打开输入流

//缓存读入数据
BufferedReader in = new BufferedReader(new InputStreamReader(file, "GBK"));
```

使用 **for** 循环按行读入一个文件。

```
String fileName = "SDIC.txt"; //文件名
InputStream file = new FileInputStream(new File(fileName)); //打开输入流

//缓存读入数据
BufferedReader in = new BufferedReader(new InputStreamReader(file, "GBK"));

for (String line = in.readLine(); line != null; line = in.readLine()) {
    System.out.println(line);
}
in.close();
```

它等价于下面这个 **while** 循环。

```
String fileName = "SDIC.txt"; //文件名
InputStream file = new FileInputStream(new File(fileName)); //打开输入流

//缓存读入数据
BufferedReader in = new BufferedReader(new InputStreamReader(file, "GBK"));
String line = in.readLine();
while (line != null) {
    System.out.println(line);
    line = in.readLine();
}
in.close();
```

通过把赋值语句写在 **while** 循环的布尔表达式里面，中间的 **while** 循环可以简写成如下所示

的代码。

```
String line;
while ((line = in.readLine()) != null) { //合并赋值语句和判断条件
    System.out.println(line);
}
```

读入的字符串在 Eclipse 控制台中显示正常不能保证读入的字符本身不是乱码。读入文件可以指定字符集编码。中文文本文件一般使用 GBK 编码。如果要把其他格式的文件转码成 GBK 编码, 可以先用记事本打开文件, 然后另存为编码是 ANSI 格式的文本文件。

读入文件时, 可以在 `InputStreamReader` 的构造方法中指定字符集。读入 GBK 编码的文本文件的代码如下所示。

```
InputStream file = new FileInputStream(new File(path));
//创建使用 GBK 字符集的 InputStreamReader
BufferedReader read = new BufferedReader(new InputStreamReader(file, "GBK"));
```

为了支持多种语言, 往往采用 UTF-8 格式编码的文件, 把文件存成 UTF-8 格式的, 然后用类似下面的代码读入。

```
String file = "D:/dict.txt";
InputStreamReader isr = new InputStreamReader(new FileInputStream(file), "UTF-8");
BufferedReader read = new BufferedReader(isr);
String line;
while ((line = read.readLine()) != null) {
    System.out.println(line);
}
```

`java.io.Writer` 用于输出字符流, `FileWriter` 类是 `Writer` 类的一个子类。使用 `FileWriter` 写入文本文件的例子如下所示。

```
String fileName = "c:/story.txt" ;

FileWriter writer = new FileWriter( fileName );
//写入四行, 可以用写字板打开这个文件
writer.write( "从前有座山, \n" );
writer.write( "山上有座庙。 \n" );
writer.write( "庙里有一个老和尚, \n" );
writer.write( "一个小和尚。 \n" );
writer.close(); //关闭文件
```



一般来说，**Writer** 是把内容立即写到硬盘。如果要多次调用 **write** 方法，则批量写入效率会更高。类似于团购，团购的价格往往比单件购买的价格低。可以使用缓存加快文件写入速度。

```
//使用缺省的缓存大小
BufferedWriter bw = new BufferedWriter(new FileWriter(fileName));
bw.write("Hello,China!"); //写入一个字符串
bw.write("\n"); //写入换行符
bw.write("Hello,World!");
bw.close(); //把缓存中的内容写入文件
```

使用 **BufferedWriter** 写入数据时，最后需要调用 **BufferedWriter** 的 **close** 方法。如果不关闭文件，可能导致缓存中的数据丢失，写入文件的数据不完整。例如，把集合中的元素写入到文件。

```
ArrayList<String> words = getLexiconEntry(); //得到词表
String fileName="C:/wordlist.txt"; //要写入的文件
BufferedWriter bw = new BufferedWriter(new FileWriter(fileName));

for(String w: words){
    bw.write(w);
    bw.write("\r\n");
}
bw.close();
```

如果要写入一个 UTF-8 编码的文本文件，则可以在 **OutputStreamWriter** 的构造方法中指定字符集。

```
File file = new File("c:/temp/test.txt"); //创建一个文件对象
BufferedWriter out =
    new BufferedWriter(new OutputStreamWriter(new FileOutputStream(file),"UTF8"));
```

按指定编码写入文本的完整代码如下所示。

```
/**
 * 向文件写入字符串
 * @param content 字符串
 * @param fileName 文件名
 * @param encoding 编码
 */
public static void writeToFile(String content,String fileName,String encoding){
    try {
        FileOutputStream fos = new FileOutputStream(fileName);
```



```

        OutputStreamWriter osw=new OutputStreamWriter(fos,encoding);
        BufferedWriter bw=new BufferedWriter(osw);
        bw.write(content);
        bw.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

如果黑板上已经有字，可以选择抹去黑板上已有的字重新写，也可以在原来的文字后继续写。如果一个文件已经存在，可以把新的内容追加写到最后，也可以从头写入新内容，也就是覆盖写。FileWriter 的构造方法区别于这两种写入方式。

```

//FileWriter 构造方法
FileWriter( String fileName, boolean mode) throws IOException

```

其中的 `mode = false` 表示覆盖写，`mode = true` 表示追加写。为了避免冲突，在一个时刻只能有一个线程写文件。

打开大的文本文件可以使用 Gvim(<http://www.vim.org/download.php>)，它是 vim 的 Windows 移植版本，或者使用收费的 UltraEdit。

## 1.15.2 二进制文件

FileWriter 只能接受字符串形式的参数，也就是说只能把内容存到文本文件。相对于文本文件，采用二进制格式的文件存储更省空间。例如，生物中的碱基可以用英文字符 A G C T 表示，也可以采用二进制格式表示，A 用 00 表示，G 用 01 表示，C 用 10 表示，T 用 11 表示。这样，二进制中的 8 位压缩成了 2 位。

读写二进制文件和文本文件使用不同的类。例如，搜索引擎中的索引库格式就是二进制文件。

InputStream 用于按字节从输入流读取数据。其中的 `int read()` 方法读取一个字节，这个字节以整数形式返回 0 到 255 之间的一个值。为什么读一个字节，而不直接返回一个 byte 类型的值？因为 byte 类型最高位是符号位，它所能表示的最大的正整数是 127。如果 `read()` 方法返回 -1，则表示已到输入流的末尾。

`InputStream` 只是一个抽象类，不能实例化。`FileInputStream` 是 `InputStream` 的子类，用于从文件中按字节读取。

```
public static void main(String[] args) throws IOException {
    String filePath = "d:/test.txt";
    File file = new File (filePath); //根据文件路径创建一个文件对象

    //如果找不到文件，会抛出 FileNotFoundException 异常
    FileInputStream fileInput = new FileInputStream(file);

    fileInput.close(); //关闭文件输入流，如果无法正常关闭，会抛出 IOException 异常
}
```

`OutputStream` 中的 `write(int b)` 方法用于按字节写出数据。`FileOutputStream` 用于按字节把数据写到文件。例如，按字节把内容从一个文件读出来，并写入另外一个新文件，也就是文件复制功能。

```
File fileIn = new File("source.txt"); //打开源文件
File fileOut = new File("target.txt"); //打开写入文件，也就是目标文件

FileInputStream streamIn = new FileInputStream(fileIn); //根据源文件构建输入流
FileOutputStream streamOut = new FileOutputStream(fileOut); //根据目标文件构建输出流

int c;
//从源文件中按字节读入数据，如果内容还没读完，则继续
while ((c = streamIn.read()) != -1) {
    streamOut.write(c); //写入目标文件
}

streamIn.close(); //关闭输入流
streamOut.close(); //关闭输出流
```

使用 `DataOutputStream` 支持直接写入整数等基本数据类型，把一个整数写入二进制文件的例子如下所示。

```
File file = new File (filePath); //根据文件路径创建一个文件对象

FileOutputStream fileOutput = new FileOutputStream(file);
BufferedOutputStream buffer = new BufferedOutputStream(fileOutput); //使用缓存写入
//将基本 Java 数据类型写到文件
DataOutputStream dataOut = new DataOutputStream(buffer);

dataOut.writeInt(nodeId); //写入整数
```

```
dataOut.close(); //关闭写入流
fileOutput.close(); //关闭文件输出流
```

使用 `DataInputStream` 把保存的整数从二进制文件读出来。

```
FileInputStream fileInput = new FileInputStream(file); //读取二进制文件
BufferedInputStream buffer = new BufferedInputStream(fileInput);
//从文件读入基本 Java 数据类型
DataInputStream dataIn = new DataInputStream(buffer);

int nodeId = dataIn.readInt(); //读出整数

dataIn.close();           //关闭读入流
buffer.close();           //关闭缓存
fileInput.close();        //关闭文件输入流
```

其中，写入整数的 `DataOutputStream.writeInt` 方法和读出整数的 `DataInputStream.readInt` 方法是对应的。写入字节数组的 `DataOutputStream.write` 方法和读出字节数组的 `DataInputStream.read` 方法也是对应的。

`DataInputStream.readByte()` 方法把最高位作为符号位，这样有可能读入负数。下面读入无符号的一个字节。

```
int type = dataIn.readUnsignedByte(); //把一个无符号的字节存入整数类型的变量
```

如果要把一个字符串保存到二进制文件，可以把字符串保存成 UTF-8 格式表示的字节数组。首先保存一个整数用来表示要读入的字节数组的长度，然后是这个字节数组。

```
byte[] by = word.getBytes("UTF-8"); //得到字符串对应的字节数组

dataOut.writeInt(by.length); //写入字节的长度

dataOut.write(by); // 写入字节数组的内容
```

读入二进制文件中的字符串，首先读入一个整数，然后是一个字节数组，最后把这个字节数组恢复成字符串。

```
int length = dataIn.readInt(); //读出字符串的长度

byte[] bytebuff = new byte[length]; //创建字节数组
int count = dataIn.read(bytebuff); //读出字节数组的内容
```



```
String word = new String(bytebuff, "UTF-8"); //根据字节数组恢复出字符串
```

把从二进制文件读入字符串封装成一个方法。

```
static String readWord(DataInputStream dataIn) throws IOException{
    int len = dataIn.readByte(); //读入长度
    byte[] bytebuff = new byte[len]; //创建字节数组
    dataIn.read(bytebuff); //读入表示字符串的字节

    return new String(bytebuff, "UTF-8"); //根据字节数组恢复出字符串
}
```

`DataInputStream.markSupported()`方法判断文件是否支持重复读入。

```
String file = "D:/test.data";
InputStream fileInput = new FileInputStream(file);
BufferedInputStream buffer = new BufferedInputStream(fileInput);
DataInputStream dataIn = new DataInputStream(buffer);

System.out.println(dataIn.markSupported());
```

例如，读入两个字节，然后回到这两个字节之前。

```
dataIn.mark(10000);
dataIn.readByte();
dataIn.readByte();
dataIn.reset();
```

用 `DataInputStream.skip` 方法跳过指定字节。

```
DataInputStream dataIn = new DataInputStream(buffer);
dataIn.skip(1103061); //跳过1103061个字节
```

用 `File.length()`方法得到文件的长度。

```
String fileName = "D:/test.doc";
File file = new File(fileName);
long length = file.length();
System.out.println("文件长度:"+length);
```

有时候需要先删除文件，例如，删除一个词典文件。

```
File dicFile = new File("./dic/" + BigramDictioanry.dataDic); //创建一个文件对象
boolean success = dicFile.delete();
```



```
System.out.println(success); //显示是否已经成功删除
```

用 `RandomAccessFile.setLength(long newLength)` 方法去掉文件尾部的若干字节。

```
RandomAccessFile file = new RandomAccessFile("f:\\down\\a.txt", "rw");

long newLength=2;
file.setLength(newLength); //去掉尾部
file.close(); //这个文件只保留了前面2个字节的内容
```

判断文件是否已经存在，如果不存在则生成这个文件。

```
File dataFile = new File(dicDir + dataDic);
if (!dataFile.exists()) {
    //如果文件不存在则写入文件
}
```

遍历路径如下所示。

```
String dirName = "D:/dir/";
File dir = new File(dirName);
File[] files = dir.listFiles();

for (int i = 0; i < files.length; i++) {
    File f = files[i];
    System.out.println(f);
}
```

用 `File.mkdirs()` 方法可以创建多级目录。例如，当一个目录不存在时，就创建它。

```
File tempDir = new File(imgPath);
if(!tempDir.exists()){
    tempDir.mkdirs();
}
```

## 1.16 多线程

人不可能像哪咤一样有三头六臂，同时做很多事情，但是机器却可以有多个运算核心。软件也可以创建很多线程并行工作。

## 1.16.1 基本的多线程

需要同时抓取不同的网站时，用多线程。

因为 Java 语言中不允许继承多个类，所以一个类一旦继承了 `Thread` 类，就不能再继承其他类了。为了避免所有线程都必须是 `Thread` 的子类，需要独立运行的类也可以继承一个系统已经定义好的叫作 `Runnable` 的接口。`Thread` 类有个构造方法 `public Thread(Runnable target)`。当线程启动时，将执行 `target` 对象的 `run()` 方法。Java 内部定义的 `Runnable` 接口很简单。

```
public interface Runnable {
    public void run();
}
```

实现这个接口，然后把要同步执行的代码写在 `run()` 方法中，测试类。

```
public class Test implements Runnable {
    public void run() {
        System.out.println("test"); //同步执行的代码
    }
}
```

运行需要同步执行的代码。

```
public class RunIt {
    public static void main(String[] args) {
        Test a = new Test();
        //a.run(); 错误的写法

        //Thread 需要一个线程对象，然后它用其中的代码向系统申请 CPU 时间片
        Thread thread=new Thread(a);
        thread.start();
    }
}
```

用不同的线程处理不同的目录页。

```
public class DownloadSina implements Runnable{
    private static int i=1;

    public void run() {
        String url = null;
        for(;;){
```

```

        synchronized(""){
            url = "http://roll.mil.news.sina.com.cn/col/zgjq/index_"+i+++".shtml";
        }

        System.out.println(url);
        //执行下载和处理目录页
    }
    System.out.println("下载完成");
}
public static void main(String[] args) throws Exception {
    DownloadSina st=new DownloadSina();
    Thread t1=new Thread(st); //启动 3 个下载线程
    Thread t2=new Thread(st);
    Thread t3=new Thread(st);

    t1.start();t2.start();t3.start();
}
}

```

在 Java 中，为了爬虫稳定性，也可以用多线程来实现爬虫。一般情况下，爬虫程序需要能在后台长期稳定运行。下载网页时，经常会出现异常。有些异常无法捕获，导致爬虫程序退出。为了主程序稳定，可以把下载程序放在子线程执行，这样即使子线程因为异常退出了，但是主线程并不会退出。测试代码如下所示。

```

public class MyThread extends Thread{
    public void run() {
        System.out.println("Throwing in " +
            "MyThread");
        throw new RuntimeException();
    }
}

public class ThreadTest {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
        try {
            Thread.sleep(2000);
        }
        catch (Exception x) {
            System.out.println("Caught it");
        }
    }
}

```



```
        System.out.println("Exiting main");  
    }  
}
```

## 1.16.2 线程池

因为删除和新建线程都是费时的工作，所以可以使用线程池 `ExecutorService` 重用线程。把要执行的任务放到线程池，让它自己调度这些任务。这些任务是轻量级线程。

有两种线程池，它们使用不同的方式创建。两种方式创建线程池。

```
int threads = 4;  
ExecutorService es = Executors.newFixedThreadPool(threads);
```

有大量任务要执行的线程池用 `newCachedThreadPool` 创建。

```
ExecutorService service = Executors.newCachedThreadPool();
```

创建一个可根据需要创建新线程的线程池，但是如果以前构造的线程可用时，就重用它们。将长期不用的线程从线程池删除，也就是删除 60 秒内没有用过的线程。就好像超市收银柜台数量是动态的，顾客多就多开几个，顾客少就少开几个收银柜台。

`newFixedThreadPool` 用于少数几个长期运行的任务的线程池。

```
ExecutorService service = Executors.newFixedThreadPool(2);
```

`newFixedThreadPool` 会重用已有的线程，但是不会创建新的线程。如果其中一个线程因为错误而结束了，将会创建一个新的接替它执行后续的任务。就好像在球场上的篮球运动员数量是固定的，如果有人受伤下场，就会有候补队员接替。

可以使用线程池执行一组任务，最简单的任务不返回值给主调线程。要返回值的任务可以实现 `Callable<T>` 接口，线程池执行任务并通过 `Future<T>` 的实例获取返回值。

`Callable` 是类似于 `Runnable` 的接口，在其中定义可以在线程池中执行的任务。`Future` 表示异步计算的结果，它提供了检查计算是否完成的方法，以等待计算的完成，并检索计算的结果。`Future` 的 `cancel` 方法表示取消任务的执行，`cancel` 方法有一个布尔参数，当参数为 `true` 时，表示立即中断任务的执行；当参数为 `false` 时，表示允许正在运行的任务运行完成。`Future` 的 `get` 方法表示等待计算完成，获取计算结果。





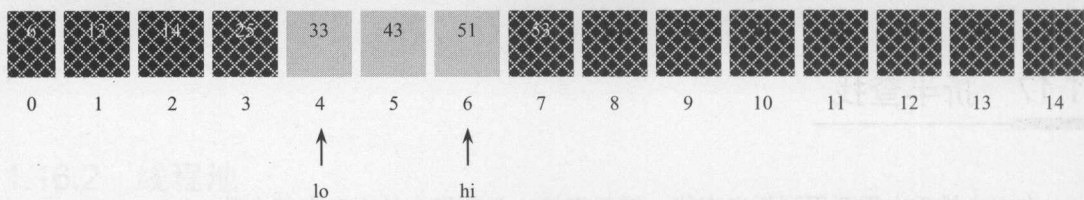


图 1-6 调整下界

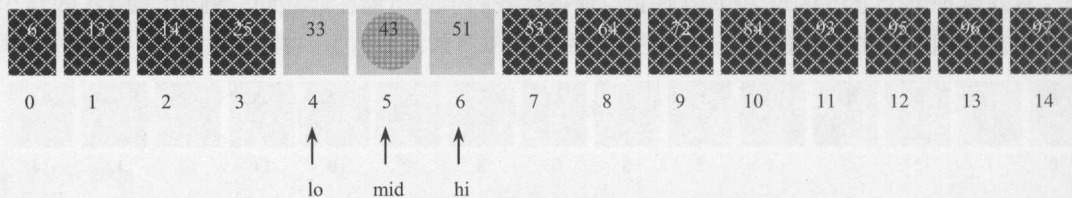


图 1-7 设置中间值

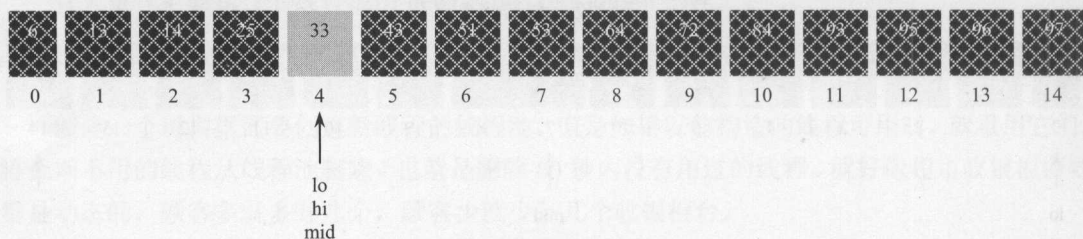


图 1-8 调整上界

首先对要查找的数据排好序，然后用递归调用的方式实现折半查找。指定一个排好序的数组和要查找的值，同时指定要查找的左边界和右边界。左边界和右边界这两个值要位于数组长度区间范围内。

```
/**
 * 寻找排好序的数组中的一个值
 *
 * @param array 要查找的数组
 * @param value 查找的值
 * @param left 左边界，这个值必须位于数组长度区间内
 * @param right 右边界，这个值必须位于数组长度区间内
 * @return 找到的值在数组中的位置，如果没找到就返回-1
 */
static int binarySearch(int[] array, int value, int left, int right) {
    if (left > right) // 退出条件
        return -1; // 没找到指定的元素
}
```

```

int middle=(left+right) >>> 1; //相当于 mid = (left+right)/2
if (array[middle]==value)
    return middle; //找到了
else if(array[middle]>value)
    return binarySearch(array,value,left,middle-1); //递归调用查找左边
else
    return binarySearch(array,value,middle+1,right); //递归调用查找右边
}

```

测试查找过程。例如，从排好序的数组{3,4,5,6,7}中找6这个数，它位于数组中第3个位置。

```

int[] data = {3,4,5,6,7};
//调用 binarySearch 方法，其中 left 的初始值是 0，right 的初始值是数组长度
int ret = binarySearch(data,6,0,data.length);
System.out.println(ret); //输出 3

```

用非递归的方法实现折半查找。

```

static int binarySearch(int[] array, int key, int fromIndex, int toIndex) {
    int low = fromIndex; //开始位置
    int high = toIndex - 1; //结束位置

    while (low <= high) {
        int mid = (low + high) >>> 1; //相当于 mid = (low + high)/2
        int midVal = array[mid]; //取中间的值

        if (midVal < key) //中间值小于要找的关键字比较
            low = mid + 1;
        else if (midVal > key) //中间值大于要找的关键字比较
            high = mid - 1;
        else
            return mid; //查找成功，返回找到的位置
    }
    return -(low + 1); //没找到，返回负值
}

```

查找区间范围越来越小。就好像在水库中捞鱼，把鱼赶到越来越小的区域，收网的时候看是否有鱼在里面。

需要注意的是：折半查找依赖于排好序的数组。如果是一个没排好序的数组，则不能使用折半查找。



折半查找的应用极其广泛，而且它的思想易于理解。第一个折半查找算法早在 1946 年就出现了，但是第一个完全正确的折半查找算法直到 1962 年才出现。

要查找动态数组中已经排好序的元素，可以直接调用 `Collections.binarySearch` 方法。

```
ArrayList<Integer> list = new ArrayList<Integer>();

//增加一些整数
list.add(3);
list.add(7);
list.add(6);
list.add(9);

//在调用 binarySearch() 方法之前，先对集合中的元素排序
//
Collections.sort(list);

//
//得到元素 2 所在的位置，如果没有找到，就返回一个负数
int index = Collections.binarySearch(list, 2);
if (index > 0) {
    System.out.println("找到的位置:" + index);
}
```

要查找数组中已经排好序的元素，可以直接调用 `Arrays.binarySearch` 方法。

```
int[] ids = {3,7,6,9}; //待查找的数组
Arrays.sort(ids);
int index = Arrays.binarySearch(ids,2);

if (index > 0) {
    System.out.println("找到的位置:" + index);
}
```

## 1.18 处理图片

`javax.imageio.ImageIO` 执行简单的读写图片文件。

```
String fileName = "D:/search/passCode2_bin.bmp";
File original_f = new File(fileName);
```



```
BufferedImage bi = ImageIO.read(original_f); //读入文件成为 BufferedImage 对象
```

ImageIO 类可以执行简单的编码和解码工作。

```
int width = 100;
int height = 200;
BufferedImage binarized = new BufferedImage(width,
        200, BufferedImage.TYPE_INT_RGB);
File file = new File(fileName);

ImageIO.write(binarized, "bmp", file);
```

一个 BufferedImage.TYPE\_BYTE\_BINARY 类型的 BufferedImage 有一个没有  $\alpha$  值的 IndexColorModel。

它代表了一个不透明的字节包装的 1 位、2 位或 4 位图像，即双色、四色和十六色图像。图像可以使用适当的颜色条目的颜色映像。如果构建一个没有传递 IndexColorModel 的 BufferedImage，则默认创建黑/白两个条目：{0, 0, 0} 和 {255, 255, 255}，但并不局限于黑色和白色，你可以从默认的 sRGB 颜色空间选择任何两种颜色，然后创建一个 IndexColorModel，再传给 BufferedImage 构造函数。

可以使用 IndexColorModel 以相同的方式创建四色和十六色图像。

```
int width = 100;
int height = 200;
BufferedImage binarized = new BufferedImage(width,
        200, BufferedImage.TYPE_BYTE_BINARY);
```

## 1.19 本章小结

本章介绍了网络爬虫所需要的 Java 技术基础。对于零基础的读者来说，通过知识的连贯整合，节省了学习 Java 基础的时间。

代码不仅仅是由人直接编写的，还可能是程序自动生成的。Eclipse 中的 ASTParser 是一个分析 Java 代码的工具包。

# 2

## 第 2 章

### 网络爬虫入门

---

大的搜索引擎，例如 Google，对整个互联网做了一个镜像。很多有专门用途的信息也需要汇总，例如网上购物或者旅游。这些专门收集互联网信息的程序叫作网络爬虫。如果把互联网比喻成一个覆盖地球的蜘蛛网，那么抓取程序就是在网上爬来爬去的蜘蛛。

虽然存在一些通用的采集器，但是因为应用目的不同，很多爬虫程序都是定制开发的。网络爬虫需要实现的基本功能包括下载网页以及遍历 URL 地址。为了高效快速地遍历大量的 URL 地址，还需要应用专门的数据结构来优化。爬虫很消耗带宽资源，设计爬虫时需要仔细考虑如何节省网络带宽。

#### 2.1 获取信息

---

在互联网普及前，人们通过报纸、杂志、电视、广播、书籍，以及面对面的交流等方式获取信息。互联网是一个巨大而有效的信息来源。网络爬虫源于人们对自动发现和获取互联网信息的需求。

写爬虫就像是从互联网中钓鱼，鱼头是下载网页，鱼身是提取有效信息，鱼尾是信息存储。先对爬虫有整体的了解，然后从头开始了解详情。

### 2.1.1 提取链接

网页和普通文档的基本区别是：它存在超级链接。需要的信息往往在一些有链接指向的网页中。超级链接也就是 a 标签，举一个 a 标签的例子。

```
<a href="http://www.lietu.com/news">新闻</a>
```

可以用正则表达式提取网页中的超级链接，但是复杂的正则表达式可读性不好。这里使用专门的 HTML 解析器来实现提取网址。

```
String url = "http://www.lietu.com/";
Document doc = Jsoup.connect(url).get(); //解析的结果就是一个文档对象
Elements links = doc.select("a[href]");
for (Element link : links) { //遍历每个链接，集合里面的每一个元素写在前面
    String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
    System.out.println(linkHref); //输出 URL 地址
}
```

### 2.1.2 采集新闻

抓取新闻列表页，得到详细页标题。首先下载并解析网页。

```
Document document = Jsoup.connect("http://politics.people.com.cn/GB/1024/").get();
System.out.println(document.html()); //打印网页源代码
```

这里的 Document 对象就是 DOM 树的根节点。

得到详细页标题所在的块，也就是 DOM 树里面的一个节点 `<ul class="list_16"> ... </ul>`。

```
Elements es = document.getElementsByClass("list_16");
```

如果下载网页出现超时，则要设置等待超时时间。

```
Jsoup.connect(url).timeout(1000).get();
```

如果这样仍然不行，则考虑用 HttpClient 来下载并提取标题和详细页链接的完整代码。



```

Document document = Jsoup.connect(
    "http://politics.people.com.cn/GB/1024/").get();
Elements es = document.getElementsByClass("list_16");

Elements links = es.select("a[href]");
for (Element link : links) { //遍历每个链接，集合里面的每一个元素写在前面
    String title = link.text();
    System.out.println(title); //输出标题
    String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
    System.out.println(linkHref); //输出 URL 地址
}

```

一个更复杂的例子，得到 <http://china.cnr.cn/yaowen/> 所有的标题和链接。

```

String url = "http://china.cnr.cn/yaowen/";
Document document = Jsoup.connect(url).timeout(1000).get();

Element content = document.getElementById("cnt1");

Elements es = content.getElementsByClass("ml20");
for (Element link : es) { //遍历每个链接，集合里面的每一个元素写在前面
    Element alink = link.getElementsByTag("a").first();
    if (alink != null) {
        System.out.println(alink.attr("href"));
        System.out.println(alink.text());
    }
}

```

## 2.2 各种网络爬虫

网络爬虫有两个基本的任务，发现包含有效信息的 URL 和下载网页。

为了获取网页，需要有一个初始的 URL 地址列表。然后通过网页中的超链接访问到其他的页面。有人可能会奇怪，像 Google 或者百度这样的搜索门户怎么设置这个初始的 URL 地址列表。一般来说，网站拥有者把网站提交给分类目录，例如，dmoz (<http://www.dmoz.org/>)，爬虫则可以从开放式分类目录 dmoz 抓取。

有的搜索引擎以浏览器访问的日志作为爬虫的初始列表。

抓取下来的网页中包含了想要的信息，一般存放在数据库或索引库等专门的存储系统中，如图 2-1 所示。如果把网页原文存储下来，就可以实现“网页快照”功能。

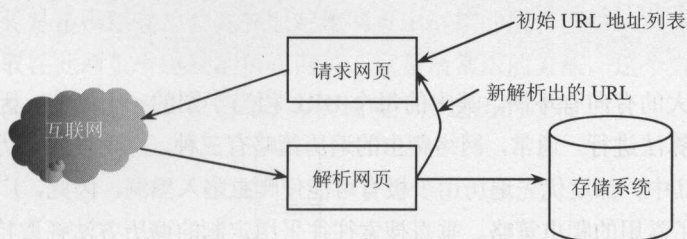


图 2-1 爬虫基本结构图

爬虫程序的工作是从一个种子链接的集合开始。把种子 URL 集合作为参数传递给网络爬虫。爬虫先把这些初始的 URL 放入 URL 工作队列（Todo 队列，又叫作 Frontier），然后遍历所有工作队列中的 URL，下载网页并把其中新发现的 URL 再次放入工作队列。为了判断一个 URL 是否已经遍历过，把所有遍历过的 URL 放入历史表。

```
HashSet<String> visited = new HashSet<String>(); //所有遍历过的 URL 放入历史表
```

爬虫抓取的基本过程如图 2-2 所示。



图 2-2 网页遍历流程图

抓取的主要流程如下。

```

while (todo.size() > 0) { //如果 Todo 队列不是空的
    //从 Todo 队列里面提取 URL
    String strUrl = todo.iterator().next();
    //下载 URL 对应的网页内容
    String content = downloadPageContent(strUrl);
    //把网页内容存储到本地
    //提取网页内容中新发现的 URL 链接
    HashSet<String> newLinks = retrieveLinks(content, new URL(strUrl));
    //把新发现的链接加入 Todo 队列
    todo.addAll(newLinks);
}
  
```

```

//从 Todo 队列里删除已经爬过的 URL
todo.remove(strUrl);
//把从 Todo 队列里删除的 URL 添加到 Visited 集合中
visited.add(strUrl);
}

```

整个互联网是一个大的有向有环图，其中的每个 URL 相当于图的一个节点，因此，网页遍历就可以采用图遍历的算法进行。通常，网络爬虫的遍历策略有三种，广度优先遍历、深度优先遍历和最佳优先遍历。其中，深度优先遍历由于极有可能使爬虫陷入黑洞，因此，广度优先遍历和最佳优先遍历就成为了常用的爬虫策略。垂直搜索往往采用定制的遍历方法抓取特定的网站。

要把坏人放到监狱里隔离一段时间，同样坏链接也要放到一个专门的集合中，在一段时间内不再抓取。否则每次都重复抓取坏链接，会浪费时间。当然坏链接过一段时间后也可能变成好链接。

## 2.2.1 信息采集器

遍历特定网站。从首页提取类别信息，然后按类别信息找到目录页。通过翻页遍历所有的目录页，提取详细页。从详细页面提取商品信息，把商品信息存入数据库。

以新浪新闻为例，同一个目录下的 URL 如下所示。

```

http://roll.news.sina.com.cn/news/gjxw/hqqw/index.shtml
http://roll.news.sina.com.cn/news/gjxw/hqqw/index_2.shtml
http://roll.news.sina.com.cn/news/gjxw/hqqw/index_3.shtml

```

以购物网站为例，同一个目录下的 URL 如下所示。

```

http://www.mcmelectronics.com/browse/Cameras/0000000002
http://www.mcmelectronics.com/browse/Cameras/0000000002/p/2
http://www.mcmelectronics.com/browse/Cameras/0000000002/p/3
http://www.mcmelectronics.com/browse/Cameras/0000000002/p/4

```

不断增加翻页参数，一直到找不到新的商品为止。

```

boolean parserIndexPage(String indexUrl){
    //解析目录页中的商品，看看是否有新的商品
    return isNew;
}

```



## 2.2.2 广度优先遍历

广度优先是指网络爬虫会先抓取起始网页中链接的所有网页，然后选择其中的一个链接网页，继续抓取在此网页中链接的所有网页。这是最常用的方法，这个方法也可以让网络爬虫并行处理，提高其抓取速度。以图 2-3 为例说明广度遍历的过程。

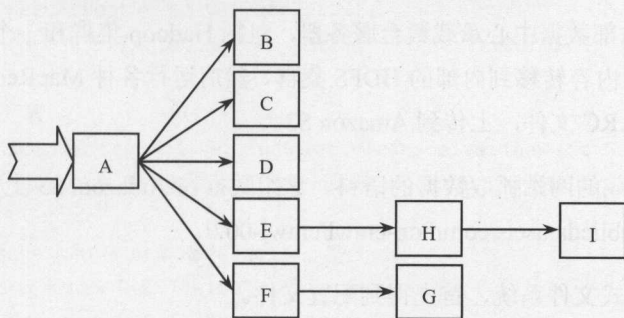


图 2-3 网络爬虫遍历的图

在图 2-3 中，A 为种子节点，首先遍历 A（第一层），然后遍历 B、C、D、E、F（第二层），接着遍历 H、G（第三层），最后遍历 I（第四层）。蜘蛛从蛛网中心往外一圈一圈地织网，可以类比作广度遍历。

广度优先遍历使用一个队列来实现 Todo 表，先访问的网页先扩展。针对图 2-3，广度优先遍历的执行过程如表 2-1 所示。

表 2-1 广度优先遍历过程表

Todo队列	Visited集合
a	null
b c d e f	a
c d e f	a b
d e f	a b c
e f	a b c d
f h	a b c d e
h g	a b c d e f
g i	a b c d e f h
i	a b c d e f h g
null	a b c d e f h g i

## 2.2.3 分布式爬虫

有两种类型的分布式爬虫：局域网内的分布式爬虫、跨局域网的爬虫。如果是跨局域网的爬虫，就有统一的服务器端分发抓取任务。客户端爬虫从服务器端取得抓取任务，下载后返回结果给服务器端。如果服务器负载太大，客户端本地可以缓存抓取结果。

commoncrawl 有一个内部数据中心承载数台服务器，包括 Hadoop 集群和一个专用爬虫库。抓取网页后，将所有下载的内容转移到内部的 HDFS 集群，然后运行各种 MapReduce 任务将要处理的内容压缩成压缩的 ARC 文件，上传到 Amazon S3。

一个由 50 亿个网页组成的网站抓取数据的语料。这组数据在 Amazon S3 上免费提供。可以通过地址获得 `s3://aws-publicdatasets/common-crawl/crawl-002/`。

写入到 Hadoop 的分布式文件系统，首先得到配置文件。

```
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
```

打开用于写文件的 `outFile`。

```
FSDataOutputStream out = fs.create(outFile);
//写完后关闭文件
out.close();
```

Apache Avro 是一个数据序列化系统，可以使用 Netty 发送和接收消息。

```
public class TestNettyServer {

    public static class MailImpl implements Mail {
        //在这个简单的例子中只返回消息的细节
        public Utf8 send(Message message) {
            return new Utf8("Sent message to [" + message.to.toString()
                + "] from [" + message.from.toString()
                + "] with body [" + message.body.toString()
                + "]);");
        }
    }

    @Test
```

```

public void test() throws Exception {
    //启动服务器
    System.out.println("starting server...");
    Responder responder = new SpecificResponder(Mail.class, new MailImpl());
    Server server = new NettyServer(responder, new InetSocketAddress(0));
    Thread.sleep(1000); //等待服务器启动

    int serverPort = server.getPort();
    System.out.println("server port : " + serverPort);

    //客户端
    Transceiver transceiver = new NettyTransceiver(new InetSocketAddress(serverPort));
    Mail proxy = (Mail)SpecificRequestor.getClient(Mail.class, transceiver);

    Message msg = new Message();
    msg.to = new Utf8("wife");
    msg.from = new Utf8("husband");
    msg.body = new Utf8("I love you!");

    try {
        Utf8 result = proxy.send(msg);
        System.out.println("Result: " + result);
        Assert.assertEquals("Sent message to [wife] from [husband] with body [I love you!]", result.toString());
    } finally {
        transceiver.close();
        server.close();
    }
}
}

```

## 2.3 爬虫相关协议

相比于人，爬虫有更快的检索速度和更深的层次，所以爬虫可能使一个站点瘫痪。

抓取网站时，为了不影响用户正常访问该网站。爬虫需要有礼貌。例如，它不会不告而访，



它会在自己的“user agent”中声称：“我是某某搜索引擎的爬虫。”

有些网站希望爬虫避免在白天对其网页进行抓取，从而不影响白天正常的对外公众服务，DNS 服务提供商也不希望大量的域名解析工作量被搜索爬虫的域名请求所占用。为了避免抓取的网站响应请求的负担过重，爬虫需要遵循礼貌性原则，不要同时发起过多的下载网页请求，这样才可能有更多的网站对爬虫友好。为了减少网站对爬虫的抱怨，建议每秒只抓取几次，把抓取任务尽量平均分配到每个时间段，并且避免高峰时段对访问的网站负担过重。

### 2.3.1 网站地图

为了方便爬虫遍历和更新网站内容，网站可以设置 Sitemap.xml。Sitemap.xml 也就是网站地图，不过这个网站地图是用 XML 写的。例如，<http://www.10010.com/Sitemap.xml>。

在网站地图中列出网站中的网址以及关于每个网址的其他元数据（上次更新的时间、更改的频率以及相对于网站上其他网址的重要程度等），以便搜索引擎抓取网站。

完整格式如下所示。

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset>
  <url>
    <loc>http://shop.10010.com</loc>
    <lastmod>2011-07-17</lastmod>
    <changefreq>daily</changefreq>
    <priority>1.0</priority>
  </url>
  <url>
    <loc>http://shop.10010.com/iphonesale/getAllIphone.action</loc>
    <lastmod>2011-07-17</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>
  ...
</urlset>
```

XML 标签的含义说明如下。

- Loc: 页面永久链接地址。

- Lastmod: 页面最后修改时间。
- Changefreq: 页面内容更新频率。
- Priority: 相对于其他页面的优先权。

### 2.3.2 Robots 协议

作为一个爬虫程序，在网上应该遵守爬虫的“道德”。何谓爬虫的“道德”？因为爬虫会访问网站，并下载其中的资源。尤其是多线程爬虫，可能会在访问一个网站的时候开启多个线程，然后使用很多 Session 进行连接，爬取网页，很容易造成网站瘫痪、不能访问等后果。还有一种情况是网站有很多东西属于自己的“秘密”，本身就不想让爬虫抓取，如果爬虫随意抓取，就相当于侵犯网站的“隐私”。

为了避免这种情况的发生，互联网行业采用了两种办法，第一种是在网站的根目录下放置一个文件，起名为 robots.txt，其中规定了哪些内容不想被抓取；另一种是设置 Robots Meta 标签。

robots.txt 文件必须放置在站点的根目录下，而且文件名必须小写。该文件包含一条或更多的记录，这些记录用空行分开（以 CR、CR/NL 或者 NL 作为结束符），在该文件中可以用#进行注解，具体使用方法和 UNIX 中的惯例一样。该文件中的记录通常以一行或多行 User-agent 开始，后面加上若干 Disallow 行，详细情况如下。

- User-agent: 该项的值用于描述搜索引擎 robots 的名字。在 robots.txt 文件中，如果有多条 User-agent 记录，说明有多个 robots 会受到该协议的限制，对该文件来说，至少要有一条 User-agent 记录。如果 User-agent 的值设为\*，则该协议对任何机器人均有效，在 robots.txt 文件中，“User-agent:\*”这样的记录只能有一条。
- Disallow: 该项的值用于描述不希望被访问到的 URL，这个 URL 可以是一条完整的路径，也可以是部分路径，任何以 Disallow 开头的 URL 均不会被 robots 访问到。例如，“Disallow:/help”对 /help.html 和 /help/index.html 都不允许搜索引擎访问，而“Disallow:/help/”则允许 robots 访问 /help.html，而不能访问 /help/index.html。

任何一条 Disallow 记录为空，说明该网站的所有部分都允许被访问，在 robots.txt 文件中，至少要有一条 Disallow 记录。如果 robots.txt 是一个空文件，则该网站对于所有的搜索引擎 Robots，都是开放的。一个 robots.txt 的例子如下所示。

```
User-agent: *  
Disallow: /cgi-bin/  
Disallow: /tmp/  
Disallow: /private/
```

以上例子表明这个网站的 Robots 协议对每个爬虫都适用, 并且不允许爬取/cgi-bin/、/tmp/和/private/下的文件。

当一个爬虫访问一个站点时, 它会首先检查该站点根目录下是否存在 robots.txt, 如果存在, 爬虫就会按照该文件中的内容来确定访问的范围; 如果该文件不存在, 所有的爬虫将能够访问网站上所有没有被口令保护的页面。

如 <http://zhidao.baidu.com/robots.txt> 规定除了百度、谷歌、msn 爬虫之外, 其他的爬虫都不被允许。

```
User-agent: Baiduspider  
Allow: /  
Disallow: /w?  
User-agent: Googlebot  
Allow: /  
User-agent: MSNBot  
Allow: /  
User-agent: *  
Disallow: /
```

抓取速度多大才文明呢? 底线是不要影响用户访问。很多小站抗压能力很差, 能不能有效的识别压力呢? 给要抓的网站做个自动压力测试。

Robots.io(<https://github.com/JamesFrost/robots.io>)是一个 Java 库, 用于解析网站的 robots.txt 文件。

RobotsParser 类提供使用 robots.io 的所有功能。

举个连接的例子, 使用 User-Agent 字符串"test"解析百度的 robots.txt。

```
RobotsParser robotsParser = new RobotsParser("test");  
robotsParser.connect("http://baidu.com");
```

如果要解析没有 User-Agent 的 robots.txt, 只需调用无参数的构造函数, 你也可以传递一个带路径的域名。



```
robotsParser.connect("http://google.com/example.htm"); //这样也有效
```

对所有方法来说, 域名参数既可以是一个字符串, 也可以是一个 URL 对象。

检查一个 URL 是否允许抓。

```
robotsParser.isAllowed("http://google.com/test"); //如果允许抓, 就返回 true
```

或者得到从文件解析出的所有规则。

```
robotsParser.getDisallowedPaths(); //返回一个字符串的数组列表
```

把解析出来的结果缓存到 robotsParser 对象, 直到再次调用 connect() 方法时, 覆盖之前解析的数据。

如果网站拒绝所有访问, 将抛出 RobotsDisallowedException 异常。

RobotsParser 返回的域名总是以正斜线结束。返回的不允许抓的路径不会以斜线开头。这样可以很容易地构造 URL。

```
//http://google.com/example.htm
robotsParser.getDomain() + robotsParser.getDisallowedPaths().get(0);
```

在 HttpClient 包中, 当执行 get 或者 post 方法时, 会默认提供对 robots.txt 的支持, 不抓取 Disallow 规定的目录下的网页。当然, HttpClient 也设置了默认选项, 可以让编写的爬虫不受 robots.txt 协议的限制, 但是, 作为一个网络爬虫作者, 我们还是提倡有“道德”的抓取, 以保障互联网行业健康发展。

现在我们讨论第二种方法, 即通过设置 Robots Meta 的值来限制爬虫的访问。这种方法是一种细粒度的方法, 能够把限制细化到每个网页。和其他的 Meta 标签(如使用的语言、页面的描述、关键词等)一样, Robots Meta 标签也放在页面的 <head></head> 中, 专门用来告诉搜索引擎 Robots 如何抓取该页的内容。

Robots Meta 标签中没有大小写之分, name=“Robots”表示所有的搜索引擎, 可以针对某个具体搜索引擎写为 name=“BaiduSpider”。Content 部分有 4 个指令选项: INDEX、NOINDEX、FOLLOW 和 NOFOLLOW, 指令间以“,”分隔。

INDEX 指令告诉搜索机器人抓取该页面。

FOLLOW 指令表示搜索机器人可以沿着该页面上的链接继续抓取下去。

Robots Meta 标签的默认值是 INDEX 和 FOLLOW，只有名为 inktomi 的标签除外，它的默认值是 INDEX 和 NOFOLLOW。

这样，一共有 4 种组合。

```
<META NAME="ROBOTS" CONTENT="INDEX, FOLLOW">  
<META NAME="ROBOTS" CONTENT="NOINDEX, FOLLOW">  
<META NAME="ROBOTS" CONTENT="INDEX, NOFOLLOW">  
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
```

其中，`<META NAME="ROBOTS" CONTENT="INDEX,FOLLOW">`可以写成`<META NAME="ROBOTS" CONTENT="ALL">`，`<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">`可以写成`<META NAME="ROBOTS" CONTENT="NONE">`。

目前看来，绝大多数的搜索引擎机器人都遵守 robots.txt 的规则，而对于 Robots Meta 标签，目前支持的并不多，但是正在逐渐增加，如著名搜索引擎 Google 就完全支持，而且 Google 还增加了一个指令 archive，可以限制 Google 是否保留网页快照。

```
<META NAME="googlebot" CONTENT="index,follow,noarchive">
```

上面的代码表示抓取该站点中的页面并沿着页面中的链接继续抓取，但是不在 Google 上保留该页面的网页快照。

## 2.4 爬虫架构

本节首先介绍爬虫的基本架构，然后介绍可以在多台服务器上运行的分布式爬虫架构。

### 2.4.1 基本架构

一般的爬虫软件，通常都包含以下几个模块。

- (1) 保存种子 URL 和待抓取的 URL 的数据结构，把网页中发现的新链接放入 Frontier 中。
- (2) 保存已经抓取的 URL 的数据结构，防止重复抓取。需要快速知道一个 URL 是否已经抓取过。

(3) 页面获取模块。

(4) 对已经获取的页面内容的各个部分进行抽取的模块。例如抽取 HTML、JavaScript 等。其他可选的模块包括下面 5 个。

(1) 负责连接前处理模块。

(2) 负责连接后处理模块。

(3) 过滤器模块。HTML 中并没有描述链接资源的类型。如果要访问某些类型的资源，往往通过后缀判断资源的类型。例如，htm 往往代表静态网页类型，img 往往代表图片。

(4) 负责多线程的模块。多线程同时下载网页。

(5) 负责分布式的模块。

各模块详细介绍如下。

### 1. 保存种子和爬取出来的 URL 的数据结构

农民会把有生长潜力的籽作为种子，我们把一些活跃的网页作为种子 URL，例如网站的首页或者列表页，因为在这些页面经常会发现新的链接。如何发现网站内的新的 URL 地址呢？抓网站中的列表页是个高效的方法。通常，爬虫都是从一系列的种子 URL 开始爬取，一般从数据库表或者配置文件中读取这些种子 URL。种子 URL 描述如表 2-2 所示。

表 2-2 最佳优先遍历过程表

字段名	字段类型	说明
Id	NUMBER(12)	唯一标识
url	Varchar(128)	网站URL
source	Varchar(128)	网站来源描述
rank	NUMBER(12)	网站PageRank值

保存待抓取的 URL 的数据结构因系统的规模、功能不同而可能采用不同的策略。一个比较小的示例爬虫程序，可能就使用内存中的一个队列，或者优先级队列进行存储。一个中等规模的爬虫程序，可能使用 BekerlyDB 内存数据库来存储，如果内存中存放不下，还可以序列化到磁盘上。但是，真正的大规模爬虫系统是通过服务器集群来存储已经爬取出来的 URL 的。并且，还会在存储 URL 的表中附带一些其他信息，比如 PageRank 值等，供之后的计算用。



## 2. 保存已经抓取过的 URL 的数据结构

已经抓取过的 URL 的规模和待抓取的 URL 的规模是一个相当的量级。正如我们前面介绍的 TODO 表和 Visited 表。它们唯一的不同是，Visited 表会经常被查询，以便确定发现的 URL 是否已经处理过。因此，如果 Visited 表数据结构是一个内存数据结构，可以采用 Hash (HashMap 或者 HashSet) 来存储，如果保存在数据库中，可以对 URL 列建立索引。

## 3. 页面获取模块

当从种子 URL 队列或者抓取出来的 URL 队列中获得 URL 后，便要根据这个 URL 来获得当前页面的内容，获得的方法非常简单，就是普通的 IO 操作。在这个模块中，仅仅是把 URL 所指的内容按照二进制的格式读出来，而不对内容做任何处理。

## 4. 提取已经获取的网页的内容中的有效信息

从页面获取模块的结果是一个表示 HTML 源代码的字符串。从这个字符串中抽取各种相关的内容是爬虫软件的目的。因此，这个模块就显得非常重要。

通常，一个网页中除了包含文本内容，还有图片、超链接等。对于文本内容，首先把 HTML 源代码的字符串保存成 HTML 文件。关于超链接提取，可以根据 HTML 语法，使用正则表达式来提取，并且把提取的超链接加入到 TODO 表中。也可以使用专门的 HTML 文档解析工具。

在网页中，超链接不仅指向 HTML 页面，还指向各种文件，除了 HTML 页面的超链接之外，其他内容的链接不能放入 TODO 表中，而要直接下载。因此，在这个模块中，还必须包含提取图片、JavaScript、PDF、DOC 等内容的一部分。在提取过程中，还要针对 HTTP 协议处理返回的状态码。本章主要研究网页的架构问题，第 3 章将详细研究从各种文件格式提取有效信息。

## 5. 负责连接前处理模块，负责连接后处理模块，过滤器模块

如果只抓取某个网站的网页，则可以对 URL 按域名过滤。

## 6. 多线程模块

爬虫主要消耗三种资源：网络带宽、中央处理器和磁盘。三者中任何一者都有可能成为瓶颈，其中网络带宽一般是租用的，所以价格相对昂贵。为了增加爬虫效率，最直接的方法就是使用多线程的方式进行处理。在爬虫系统中，要处理的 URL 队列往往是唯一的，多个线程顺序要从队列中取得 URL，然后各自进行处理（处理阶段是并发进行的）。通常，可以利用线程池来管理线程。程序中最大可以使用的线程数是可配置的。

## 7. 分布式处理

分布式是当今计算的主流，这项技术也可以用在网络爬虫上面。2.4.2 节介绍多台机器并行采集的方法。

### 2.4.2 分布式爬虫架构

把抓取任务分布到不同的节点，分布主要是为了可扩展性，也可以使用物理分布的爬虫系统，让每个爬虫节点抓取靠近它的网站。例如，北京的爬虫节点抓取北京的网站，上海的爬虫节点抓取上海的网站。又如，电信网络中的爬虫节点抓取托管在电信网络中的网站，联通网络中的爬虫节点抓取托管在联通网络中的网站。

分布式爬虫可以使用有中央服务器的方式实现，也可以用无中央服务器的方式实现。图 2-4 是一种没有中央服务器的分布式爬虫结构。

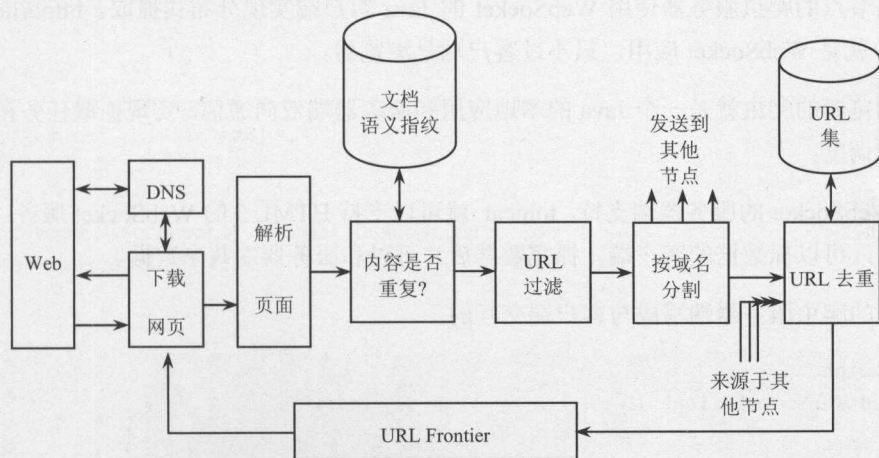


图 2-4 分布式爬虫架构图

要点在于按域名分配采集任务。每台机器扫描到的网址，不属于它自己的会交换给属于它的机器。例如，有一台机器专门抓取 s 开头的网站 <http://www.sina.com.cn> 和 <http://www.sohu.com>，而另外一台机器专门抓取 q 开头的网站 <http://www.qq.com>。为了防止单点失败，可以两台机器互为镜像，共同处理一个任务。系统自动选出的主控服务器记录了每台爬虫机器的工作任务，如果某台爬虫机器宕机了，主控服务器就把这台机器的抓取任务分配给其他机器。

垂直信息分布式抓取的基本设计。

(1) 要处理的信息根据首字母散列到 26 个不同的爬虫服务器中，让不同的机器抓取不同的信息。

(2) 每台机器通过配置文件读取自己要处理的字母。每台机器抓取完一条结果后把该结果写入到统一的数据库中。例如，有 26 台机器，第一台机器抓取字母 a 开头的公司，第二台机器抓取字母 b 开头的公司，依此类推。

(3) 如果某一台机器抓取速度太慢，则把该任务拆分到其他的机器上。

可以把检测重复内容的功能专门交给一个集群处理。解析页面的功能和下载网页的功能可以直接在同一台机器上实现。

可以把每台爬虫机器都设置成代理服务器，仍然由一台机器发起请求，但是实际抓取分配一些到代理机器执行。

有中央节点的爬虫服务器使用 WebSocket 的 Java 客户端实现分布式抓取。<http://lietu.com/bot/bot.html> 就是 WebSocket 应用，只不过客户端是浏览器。

不用浏览器的爬虫就是一个 Java 的本地应用和服务器端双向通信，实现抓取任务在互联网上的分布式调度。

如果 WebSocket 的服务器端支持，tomcat 就可以支持 HTML 5 的 WebSocket 服务。要做个爬虫客户端，可以抓数据的客户端。抓完数据后，可以和服务器端共享数据。

把自己的爬虫服务器端写成与客户端交互的。

```
@ClientEndpoint
public class WebSocketClient {

    protected WebSocketContainer container;
    protected Session userSession = null;

    public WebSocketClient() {
        container = ContainerProvider.getWebSocketContainer();
    }

    public void Connect(String sServer) {

        try {
            userSession = container.connectToServer(this, new URI(sServer));
        }
    }
}
```



```

        } catch (DeploymentException | URISyntaxException | IOException e) {
            e.printStackTrace();
        }

    }

    public void SendMessage(String sMsg) throws IOException {
        userSession.getBasicRemote().sendText(sMsg);
    }

    @OnOpen
    public void onOpen(Session session) {
        System.out.println("Connected");
    }

    @OnClose
    public void onClose(Session session, CloseReason closeReason) {
    }

    @OnMessage
    public void onMessage(Session session, String msg) {
        System.out.println(msg);
    }

    public void Disconnect() throws IOException {
        userSession.close();
    }
}

```

Running the client

```

import java.io.IOException;

public class TestWS {

    public WebSocketClient wsc;
    public TestWS() {
    }

    public void Start() throws InterruptedException, IOException {

        wsc = new WebSocketClient();
    }
}

```

```

        wsc.callback = this;
        wsc.Connect("ws://192.168.0.25:9000/"); //required a '/' on the end of it when being
run from a java client.
        Thread.sleep(1000);
        wsc.Disconnect();
    }

    public static void main(String[] args) throws InterruptedException, IOException
    {
        TestWS t = new TestWS();
        t.Start();
        Thread.sleep(1000);
    }
}

```

### 2.4.3 垂直爬虫架构

垂直爬虫往往抓取指定网站的新闻或论坛等信息。

可以指定初始抓取的首页或者列表页，然后提取相关的详细页中的有效信息存入数据库，总体结构如图 2-5 所示。

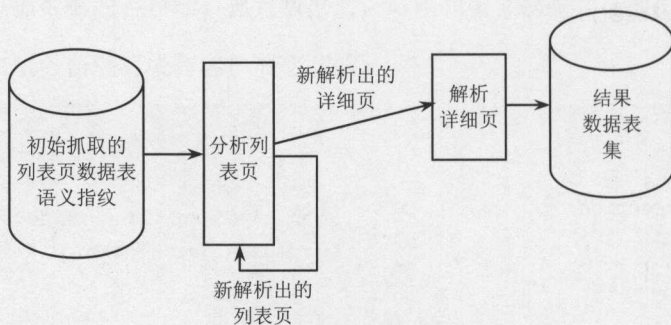


图 2-5 垂直爬虫结构图

垂直爬虫涉及的功能如下。

- 从首页提取不同栏目的列表页。

- 网页分类：把网页分类成列表页、详细页或者未知类型。
- 列表页链接提取：从列表页提取同一个栏目下的列表页，这些页面往往用“下一页”“尾页”等信息描述。往往有个参数来控制翻页，这个参数叫作翻页参数，例如，`http://...&_pgn=2` 中的 `_pgn`。有时候，列表页会显示总共有多少页，可以根据翻页参数列出所有的列表页。如果没有，还可以根据翻页参数一直往后翻页，直到找不到新信息或者出错为止。
- 详细页面内容提取：从详细页提取网页标题、主要内容、发布时间等信息。一般把每行数据封装到一个对象中。

每个网站用一个线程抓取方便控制对被抓网站的访问频率。最好有通用的信息提取方式来解析网页，这样可以减少人工维护成本。同时，也可以采用专门的提取类来处理数据量大的网站，从而提高抓取效率。

解析详细页面成为结构化数据并存储结果的工作可以交给专门的机器去做。所以可以把下载的详细页面放入消息队列，然后由解析详细页面的机器从消息队列中取出要分析的详细页面。

## 2.5 自己写网络爬虫

通过一个种子网页，遍历到感兴趣的网页，往往用 HTML 解析器从网页源代码提取网址。Jsoup 就是一个常用的 HTML 解析器。

```
String url = "http://www.lietu.com/";
Document doc = Jsoup.connect(url).get(); //解析的结果就是一个文档对象
Elements links = doc.select("a[href]");
for (Element link : links) { //遍历每个链接，集合里面的每一个元素写在前面
    String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
    String linkText = link.text(); //得到锚点上的文字说明
    System.out.println(linkHref+" "+linkText); //输出 URL 地址和锚点上的文字说明
}
```

根据 URL 地址或者锚文本来决定是否往下遍历这个页面。

互联网可以看成是一个有回路的图。遍历树用一个队列就可以，遍历有回路的图还需要避免重复访问。



```

Page currentNode = rootPage; //从根节点开始遍历网站

Deque<Page> queueNode = new ArrayDeque<Page>(); //存放节点数据的队列
queueNode.addFirst(currentNode);

while (!queueNode.isEmpty()) { //广度优先遍历所有树节点, 将其加入至队列中
    currentNode = queueNode.pollFirst(); //取出并删除队列中第一个节点

    //处理当前节点的子节点
    if (currentNode.childrenNode != null) {
        queueNode.addLast(currentNode.childrenNode);
    }
}

```

从一个已知的种子链接开始宽度遍历网页, 为了知道还有哪些网址可以抓, 需要把新发现的网址放入 URL 工作队列中, 使用 `ArrayDeque` 实现这个工作队列。为了避免重复访问一个网址, 需要把所有遍历过的 URL 放入历史表, 使用 `HashSet` 实现这个历史表。

首先定义相关数据结构。

```

static ArrayDeque<String> todo = new ArrayDeque<String>(); //要遍历的网址
static HashSet<String> visited = new HashSet<String>(); //已经访问过的 URL

```

然后从一个网址开始遍历。

```

todo.add("http://www.lietu.com"); //把种子 URL 加入到工作队列
while (!todo.isEmpty()) { //如果 Todo 队列不是空的
    //从 Todo 队列里面提取第一个 URL, 并从 Todo 队列删除这个 URL
    String strUrl = todo.pollFirst();
    //下载 URL 对应的网页内容
    String content = downloadPageContent(strUrl);
    //提取没有访问过的 URL 链接
    HashSet<String> newLinks = retrieveLinks(content, new URL(strUrl));
    //把新发现的 URL 加到 Todo 队列最后
    todo.addAll(newLinks);
    //把访问过的 URL 添加到 Visited 集合中
    visited.add(strUrl);
}

```

如果采用 `Queue` 来实现 `Todo`, 则对每个增加到 `Todo` 的元素都需要用对象封装。因为 `ArrayDeque` 底层采用数组实现, 所以增加到 `ArrayDeque` 的元素不需要用对象封装。 `ArrayDeque`

性能比 Queue 更好，所以 todo 采用 ArrayDeque 实现。

Visited 集合增长迅速，内存很快就会放不下，所以要采用更节省内存的数据结构实现 Visited 集合，或者借助外部存储。

如何快速抓取到有效的信息是一个值得研究的问题。一个网页的内容很大程度上取决于别的网页如何描述这个网页。锚点文字是另外一个网页对当前网页的描述。例如，在 “<a href=“lietu.com”>猎兔搜索</a>” 中，锚点文字是“猎兔搜索”，使用 Jsoup 提取锚点文字代码如下。

```
String html = "out<a href=\"lietu.com\">猎兔搜索</a>out";
Document doc = Jsoup.parse(html);

Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
    String anchor = link.ownText(); //锚点文字
    System.out.println(linkHref + " : " + anchor);
}
```

## 2.6 URL 地址查新

在科技论文发表时，为了避免重复研究和抄袭，需要到专门的科技情报所做论文查新。为了避免重复抓取，URL 地址也需要查新，判断解析出的 URL 是否已经遍历过，也叫 URLSeen 测试。URLSeen 测试对爬虫性能有重要的影响。本节介绍两种实现快速 URLSeen 测试的方法。

在介绍爬虫架构的时候，我们讲解了 Frontier 组件的作用。它作为一个基础的组件，为爬虫提供 URL。因此，在 Frontier 中有一个数据结构来存储 URL。在一些小的爬虫程序中，使用内存队列（ArrayList、HashMap 或 Queue）或者优先级队列来存储 URL，但内存是有限的。在商业应用中，URL 地址数据量非常大。早期的爬虫经常把 URL 地址放在数据库表中，但数据库对于这种简单的结构化存储来说效率太低。因此，考虑使用嵌入式数据库来存储。这里的嵌入式的意思和硬件无关，是嵌入在程序中的意思。

## 2.6.1 嵌入式数据库

把内存中放不下的数据存入 B 树中，如图 2-6 所示。

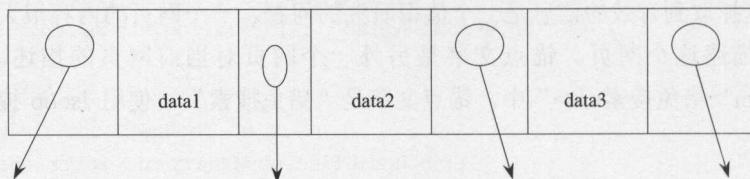


图 2-6 B 树中的节点

B 树的根节点位于内存，而叶节点位于硬盘的文件中。

嵌入式数据库 Berkeley DB（简称 BDB）正是 B 树的现成实现。BDB 中的一个数据库只能存储一些键/值对，也就是键和值两列。

Berkeley DB (<http://www.oracle.com/database/berkeley-db/index.html>)，是一个嵌入式数据库。嵌入式数据库的意思不是说它只应用在嵌入式系统上，而是不需要通过 JDBC 访问数据库，也不单独启动进程来管理数据。Berkeley DB 中的一个数据库只能存储一些键/值对，也就是键和值两列。Berkeley DB 底层实现采用 B 树，可以把它看成可以存储大量数据的 HashMap。Berkeley DB 的 C++ 版本首先出现，在此基础上又实现了 Java 本地版本。但本书中只用到 Java 版本，可以用它来存储已经抓取过的 URL 地址。如果使用 Berkeley DB Java Edition 4.0.103，那么在 Eclipse 项目中只需要导入一个 jar 包——je-4.0.103.jar。在 Berkeley DB 中，数据库存储在环境变量中，所以首先要新建环境变量。

```
EnvironmentConfig envConfig = new EnvironmentConfig();
envConfig.setTransactional(false); //不需要事务功能
envConfig.setAllowCreate(true); //允许创建新的数据库文件
exampleEnv = new Environment(envDir, envConfig);
```

释放环境变量的方法。

```
exampleEnv.sync(); //同步内存中的数据到硬盘
exampleEnv.close(); //关闭环境变量
exampleEnv = null;
```

创建数据库，键是字符串，值是一个类。



```
String databaseName= "ToDoTaskList.db";
DatabaseConfig dbConfig = new DatabaseConfig();
dbConfig.setAllowCreate(true);    //允许创建新的数据库文件
dbConfig.setTransactional(false); //不需要事务功能

//打开用来存储类信息的数据库
dbConfig.setSortedDuplicates(false);
//用来存储类信息的数据库不要求能够存储重复的关键字
Database myClassDb = exampleEnv.openDatabase(null, "classDb", dbConfig);
//初始化用来存储序列化对象的 catalog 类
catalog = new StoredClassCatalog(myClassDb);
TupleBinding keyBinding =
    TupleBinding.getPrimitiveBinding(String.class);
//把 value 作为对象的序列化方式存储
SerialBinding valueBinding = new SerialBinding(catalog, NewsSource.class);
store = exampleEnv.openDatabase(null, databaseName, dbConfig);
store.close();//关闭存储数据库
myClassDb.close();//关闭元数据库
```

建立数据的映射。

```
//创建数据存储的映射视图
this.map = new StoredSortedMap(store, keyBinding, valueBinding, true);
```

然后可以像操作普通的哈希表一样，对 `StoredSortedMap` 用 `put` 方法写入数据，`get` 方法读出数据，通过迭代器遍历访问。但是普通的哈希表只能把所有的数据存储在内存中，而 `StoredSortedMap` 则会通过内外存交换支持更多的数据存取，完整的例子如下所示。

```
String dir = "./db/";
EnvironmentConfig envConfig = new EnvironmentConfig();
envConfig.setTransactional(false);
envConfig.setAllowCreate(true);

Environment env = new Environment(new File(dir), envConfig);

//使用一个通用的数据库配置
DatabaseConfig dbConfig = new DatabaseConfig();
dbConfig.setTransactional(false);

dbConfig.setAllowCreate(true);
//如果有序列化绑定则需要类别数据库
Database catalogDb = env.openDatabase(null, "catalog", dbConfig);
```

```

ClassCatalog catalog = new StoredClassCatalog(catalogDb);

//关键字数据类型是字符串
TupleBinding<String> keyBinding =
    TupleBinding.getPrimitiveBinding(String.class);

//值数据类型也是字符串
SerialBinding<String> dataBinding =
    new SerialBinding<String>(catalog, String.class);

Database db = env.openDatabase(null, "url", dbConfig);

//创建一个映射
SortedMap<String, String> map = new StoredSortedMap<String, String>
    (db, keyBinding, dataBinding, true);
//把抓取过的 URL 地址作为关键字放入映射
String url = "http://www.lietu.com";
map.put(url, null);
if (map.containsKey(url)){
    System.out.println("已抓取");
}

```

为了能正确处理大量数据，需要增加 Java 虚拟机运行的内存使用量，否则会出现内存溢出的错误。例如增加最大内存用量到 800M，可以使用虚拟机参数“-Xmx800m”。

## 2.6.2 布隆过滤器

判断 URL 地址是否已经抓取过还可以借助于布隆过滤器（Bloom Filter）。布隆过滤器的实现方法是：利用内存中的一个长度是  $m$  的位数组  $B$ ，对其中所有位都置 0，如图 2-7 所示。

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

图 2-7 位数组  $B$  的初始状态

然后根据  $k$  个不同的散列函数，对每个遍历过的 URL 执行散列，每次散列的结果都是不大于  $m$  的一个整数  $a$ 。根据散列得到的数在位数组  $B$  对应的位上置 1，也就是让  $B[a]=1$ 。图 2-8 显示了放入 3 个 URL 后位数组  $B$  的状态，这里  $k=3$ 。

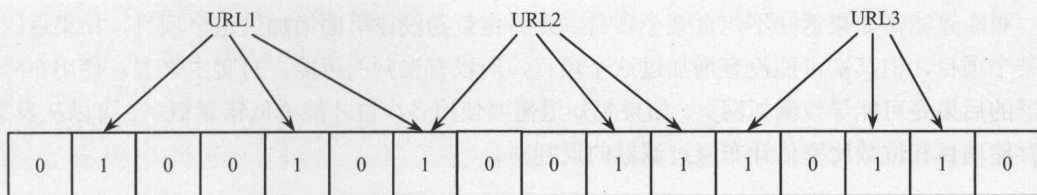


图 2-8 放入数据后位数组 B 的状态

每次插入一个 URL，也执行  $k$  次散列，只有当全部位都已经置 1 了才认为这个 URL 已经遍历过。

### 2.6.3 实现布隆过滤器

用 Java 内部的 BitSet 类表示位数组 B。

```
public class BloomFilter<E> implements Serializable {
    private BitSet bitset; //存储 URL 地址的位数组
}
```

B[a]=1 的实现代码如下所示。

```
bitset.set(a, true);
```

如下是使用布隆过滤器 (<http://code.google.com/p/java-bloomfilter/>) 的一个例子。

```
//创建一个 100 位的布隆过滤器，优化成包含 4 个项目
BloomFilter<String> urlSeen = new BloomFilter<String>(100, 4);
//增加内容到布隆过滤器
urlSeen.add("www.lietu.com");
urlSeen.add("www.sina.com");
urlSeen.add("www.qq.com");
urlSeen.add("www.sohu.com");
//测试布隆过滤器是否记得这个项目
if (urlSeen.contains("www.lietu.com")) {
    System.out.println("已经存在的概率 "
        + (1 - urlSeen.expectedFalsePositiveProbability()));
} else {
    System.out.println("一定不存在");
}
```



布隆过滤器如果返回不包含某个项目，那肯定就是没往里面增加过这个项目，如果返回包含某个项目，但其实可能没有增加过这个项目，所以有误判的可能。对爬虫来说，使用布隆过滤器的后果是可能导致漏抓网页。如果想知道需要使用多少位才能降低错误概率，可以从表 2-3 的存储项目和位数比率估计布隆过滤器的误判率。

表 2-3 布隆过滤器误判率表

比率 (items:bits)	误判率
1:1	0.63212055882856
1:2	0.39957640089373
1:4	0.14689159766038
1:8	0.02157714146322
1:16	0.00046557303372
1:32	0.00000021167340
1:64	0.00000000000004

为每个 URL 分配两个字节就可以达到千分之几的冲突。例如，一个比较保守的实现，为每个 URL 分配 4 个字节，项目和位数比是 1:32，误判率是 0.00000021167340。对于 5 000 万数量级的 URL，布隆过滤器只占用了 200M 的空间，并且排重速度超快，一遍下来不到两分钟。布隆过滤器并不像 HashMap 那样存键对象，它只是个数组而已，所以很省空间。

在 SimpleBloomFilter 中实现的把对象映射到位集合的方法。

```
public void add(E element) throws UnsupportedOperationException {
    long hash;
    String valString = element.toString();
    for (int x = 0; x < k; x++) {
        hash = createHash(valString + Integer.toString(x));
        hash = hash % (long)bitSetSize;
        bitset.set(Math.abs((int)hash), true);
    }
}
```

这个实现方法计算了  $k$  个相互独立的散列值，因此误判率较低。

为了支持重启后运行，把布隆过滤器的状态保存到文件中。

```
public void saveBit(String filename) {
    File file = new File(filename);
    ObjectOutputStream oos =
```

```

        new ObjectOutputStream(new FileOutputStream(file, false));
oos.writeObject(bitSet);
oos.flush();
oos.close();
}

```

如下的代码把布隆过滤器的状态从先前保存的文件中读出来。

```

public void readBit(String filename) {
    File file = new File(filename);
    if (!file.exists()) {
        return;
    }
    bitSet.clear();
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file));
    bitSet = (BitSet)ois.readObject();
    ois.close();
}

```

## 2.7 部署爬虫

为了抓取更加稳定，往往让爬虫运行在一个独立的控制台进程中，可以在 MANIFEST.MF 文件中声明要运行的类。例如，假设 `com.lietu.crawler.Spider` 包含 `main` 方法。

```

Manifest-Version: 1.0
Sealed: true
Main-Class: com.lietu.crawler.Spider
Class-Path: nekohtml.jar lucene-core-3.0.2.jar .

```

其中的 `Class-Path` 声明了依赖的 `jar` 包，最后的点代表当前路径。

通过 `Ant` 执行的 `build.xml` 来自动生成可执行的 `jar` 包。`Ant` 通过调用 `target` 树，就可以执行各种 `task`。每个 `task` 实现了特定接口对象。由于 `Ant` 是 XML 格式的文件，所以很容易维护和书写，而且结构很清晰，`Ant` 可以集成到开发环境中。由于 `Ant` 的跨平台性和操作简单的特点，`Eclipse` 已经集成了 `Ant`。

`crawler.jar` 包里面要正好包含有用的 `class` 文件，既不能包含测试部分代码，也不能包含源

文件。如果 Java 源代码文件编码不一致可能会出错，可以把编码统一成 GBK 或者 UTF-8。例如，采用 UTF-8 编码，则 build.xml 的写法如下所示。

```
<javac encoding="utf-8" debug="true" srcdir="${src}" destdir="${bin}"
classpathref="project.class.path" target="1.6" source="1.6"/>
```

完成打包后执行 crawler.jar。

```
>java -jar crawler.jar
```

批处理文件。

```
@ECHO OFF
set CLASSPATH=.
set
CLASSPATH=%CLASSPATH%;sktrade.jar;commons-io-2.4.jar;commons-logging-1.1.1.jar;httpclien
t-4.2.1.jar;httpcore-4.2.1.jar;jsoup-1.6.2.jar;mysql-connector-java-5.0.8-bin.jar

java -Xms1024m -Xnoclassgc com.lietu.show.DownProducts
```

## 2.7.1 部署到 Windows

写一个批处理文件从命令行运行爬虫程序，用逗号分隔 jar 文件。例如，要执行 com.lietu.crawler.Spider 类，spider.bat 文件的内容如下所示。

```
java -Xmx900m -cp ./lib/spider.jar;. com.lietu.crawler.Spider
```

## 2.7.2 部署到 Linux

在 Linux 下设置类路径的方法与 Windows 不同，使用分号分隔 jar 文件，而不是逗号。

```
java -Xmx900m -cp ./*:.. com.lietu.crawler.Spider
```

安装 lrzsz 之后可以使用 rz 命令上传文件到爬虫所在的 Linux 服务器。

```
#yum install lrzsz
```

或者使用 Bitwise SSH Client 中的 SFTP 窗口上传文件，运行 spider.sh。

```
# cat ./spider.sh
```



```
nohup java -Xmx1000m -jar crawler.jar &  
    把 spider.sh 变成可执行的。  
#chmod +x ./spider.sh
```

## 2.8 本章小结

有人说“知识就是力量”，爬虫就是自动获取知识的第一步。在互联网出现以前，大量收集信息是很困难的事情。公元前，埃及的托勒密一世有一个梦想：要把世界上所有的文献、所有值得记下来的东西全部收藏。托勒密一世和他的后代托勒密二世、托勒密三世在港口城市亚历山大兴建图书馆，雇人抄写图书。所有到亚历山大港的船只都要把携带的书交来检验，如发现现有图书馆没有的书，则马上抄录，留下原件，将复制件奉还原主。埃及人还采用了重金收购等方法，最终建立起了当时世界上最大的图书馆。在这个图书馆建立以前，知识在很大程度上是区域性的，自建成第一座世界性图书馆后，知识也随之成为世界性的了。

没做过爬虫的人可能以为爬虫只是一个广度有限遍历的应用，做过爬虫的可能就不这么想了。本章首先熟悉了网络爬虫工作的基本原理，然后了解抓取网页的实现。网络爬虫在某种程度上可以看成是一个 HTTP 客户端应用。首先介绍和网站打交道所使用的 HTTP 协议以及 TCP/IP 协议，然后用最简单的方法下载网页，最后介绍实际下载网页所使用的 HTTPClient。

除了网页类的文本信息，还有图片、FTP 等，本章还介绍了比普通网页更规范的 RSS 种子的下载方法，而对于网页中的 JavaScript 的处理是难点。此外，还介绍了 Web 图的存取和挖掘算法。

可以用布隆过滤器或者 Berkeley DB 实现 URL 地址查新。还可以使用 Berkeley DB 实现网页快照功能，也就是根据 URL 地址查询网页原文。分布式 URL 地址查新可以使用 Redis。

接下来简单回顾下网络爬虫的开发历史。

1994 年，休斯敦大学的 Eichmann (<http://slis.uiowa.edu/eichmann/index.shtml>) 在美国航空航天局的资助下开发了互联网爬虫 RBSE (Repository Based Software Engineering)。RBSE 将爬虫和索引程序分离，这样不需要重新抓取就可以更新索引。爬虫执行宽度优先遍历或者有限深度优先的遍历。Spider 程序创建和操作存储在 Oracle 数据库中的 Web 图。一个修改后的 ASCII 字符浏览器 (mite) 根据 URL 下载指定的网页存储到本地文件，并把这个网页中的链接提取出来。

1994 年, 华盛顿大学计算机系的学生 Pinkerton 开发了分布式爬虫 WebCrawler。运行在多个机器上的爬虫代理 (Agent) 用 HTTP 协议下载网页并把 HTML 格式的网页解析成纯文本。WebCrawler 使用简单的广度优先遍历方法抓取部分互联网中的网页。因为早期开源数据库不稳定, 为了避免数据库服务器崩溃, 它使用商业的 Oracle 数据库作为存储核心。1997 年 Excite 收购了 WebCrawler。Pinkerton 后来担任为 Lucene 和 Solr 提供商业服务的 Lucid Imagination 公司的首席架构师。

1998 年, 斯坦福大学的学生 Brin 和 Page 用 Python 开发了分布式爬虫系统 Google Crawler。URL 服务器给几个爬虫程序发送要抓取的 URL 列表。在解析文本的时候, 把新发现的 URL 传送给 URL 服务器并检测这个 URL 是不是已经存在, 如果不存在, 就把该 URL 加入到 URL 服务器。爬虫程序使用了 DNS 缓存来提高 DNS 查询效率。Google Crawler 对谷歌搜索的成功有重大的作用。

1999 年, Compaq 公司的 Heydon 和 Najork 开发了 Mercator。C 语言的网络 IO 速度比 Java 快, 因此直到今天很多在线运行的爬虫仍然采用 C 或 C++ 开发, 但 Mercator 却是一个用 Java 成功实现的网络爬虫。Mercator 是分布式模块化的。Mercator 用作 Alta Vista 搜索引擎的网络爬虫。Mercator 的模块包括可互换的“协议模块”和“处理模块”。“协议模块”负责怎样获取网页 (例如, 使用 HTTP 协议), “处理模块”负责怎样处理页面。标准处理模块仅仅包括了解析页面和抽取新的 URL, 可以用其他处理模块来索引网页中的文本, 或者搜集 Web 统计数据。Mercator 使用 64 位的文档语义指纹来判断网页内容是否重复。因为布隆过滤器存在误判的情况, Mercator 没有采用 Internet Archive 爬虫中曾经采用的布隆过滤器。Mercator 用带缓存的 checksum 来判断 URL 是否抓取过。

2001 年, IBM 的 Almaden 研究中心的 Edwards 等人开发了一个与 Mercator 类似的分布式模块化的爬虫 WebFountain。它的特点是一个“控制者”机器控制一系列的“蚂蚁”机器, 可以实现增量抓取。经过多次下载页面后, 可以推测出每个页面的变化率, 然后可以获得一个最大的新鲜度的访问策略。商业信息网站 Factiva 使用 WebFountain 收集企业信用信息。WebFountain 是使用 C++ 实现的。

2002 年, FAST 公司的 Risvik 和 Michelsen 开发了分布式爬虫 FAST Crawler, 在 Fast Search&Transfer 和 www.AllTheWeb.com 网站使用。节点之间通过 distributor 模块交换发现的链接。每个机器有一个“文档调度器”来维护要下载的文档队列。“文档处理器”下载完网页后把网页存储在本地存储子系统。FAST Crawler 实现了增量式抓取, 优先抓更新活跃的网页。2004 年, Yahoo 收购 AllTheWeb 网站。

Cho 和 Garcia-Molina 在论文 *Parallel Crawlers* 中研究了如何设计有效的并行爬虫, 在论文 *Effective Page Refresh Policies For Web Crawlers* 中研究了网页更新方法。

2003 年初, 为了对网上的资源进行归档, 建立网络数字图书馆, 开发了开源网络爬虫 Heritrix (<https://webarchive.jira.com/wiki/display/Heritrix/Heritrix>)。这个系统以运行时高可配置性的模块式开发, 所以非常适合做实验。

早期的互联网搜索研究中有很多是关于主题爬虫的。Menczer 和 Belew 在论文 *Adaptive Information Agents in Distributed Textual Environments* 中设想了一种为个人用户服务的, 由自主软件代理的主题爬虫。用户同时输入网址和关键字, 代理就会尝试寻找对用户有用的网页, 而且用户也可以对这些网页进行评估并反馈给系统。

Chakrabarti 在论文 *Focused Crawling: a New Approach to Topic-Specific Web Resource Discovery* 中主要研究了主题爬虫, 他们的爬虫使用分类来判断抓取的网页的主题。他们认为对于提供主题链接方面来说, 非主题爬虫无法与主题爬虫相比较。主题爬虫可以成功截获主题, 广泛的网络连接结构使非主题爬虫迅速地移动到其他主题上。

Unicode 规范是一项难以置信的复杂的工作, 包含了上万的字符。因为一些非西方语言特性的原因, 许多象形文字都是一组 Unicode 字符组成的。所以这个规范的细节不仅说明这些字是什么, 而且解释了它们是如何组合的, 新的字符仍然在源源不断地加入到 Unicode 中。

Bergman 的论文 *The Deep Web: Surfacing Hidden Value* 是对深网的一个深入的研究。尽管这项研究和 Web 标准一样的古老, 它向我们展示了如何通过搜索引擎进行抽样才能对在网络中的量指数挂钩有帮助。这个研究估算出约有 5 500 亿个网页存在于深网中, 却只有 10 亿个网页出现在可见网络中。他在一个近期的研究调查中表明, 深网近几年一直处于持续迅速地发展状态中。于是一种由 Ipeirotis 与 Gravano 所提出来的叫作查询探测的用于对深网的数据库进行探测分析的模型就出现了。抓取暗网可以参考一些 Web 应用自动测试工具。

网站地图、robots.txt 文件、RSS feeds 和 Atom feeds 都有它们自己的规范。这些格式说明那些成功的 Web 标准一般都是很简单的。

对于一些应用程序来说, 可以用数据库系统来存储爬虫下载的文件。这方面有一些教科书, 例如 Garcia-Molina 的著作 *Database Systems: the Complete Book* 提供了许多数据库如何运行的信息, 也包含一些重要功能, 如查询语句、锁、恢复等。Chang 在论文《Bigtable: 一个结构化数据的分布式存储系统》中描述了 Bigtable。



另外大型互联网公司出于类似的目的：大规模分布、高吞吐量、不需要昂贵的查询语句或者详细的事务支持，纷纷建立了自己的数据库系统。DeCandia 在论文 *Dynamo: Amazon's Highly Available Key-value Store* 中描述了亚马逊公司的 Dynamo 系统，它拥有低延迟的保证。Yahoo 使用 UDB 系统处理巨大的数据量。

DEFLATE 和 LZW 是 2 个文本压缩工具。DEFLATE 是现在流行的 zip、gzip 和 zlib 这些压缩工具的基础。LZW 是 Unix 压缩命令的基础，同样也适用于 GIF、Postscript，以及 PDF 等文件形式。Witten 写的 *Managing Gigabytes: Compressing and Indexing Documents and Images* 提供了一些关于文本和图片压缩算法的详细讨论。

Yu 的论文 *Improving Pseudo-Relevance Feedback in Web Information Retrieval Using Web Page Segmentation* 和 Gupta 的论文 *DOM-based Content Extraction of HTML Documents* 是对从网页中提取正文非常有用的文献。基于内容的网页排重将在后续章节中介绍。

# 3

## 第 3 章

---

### 定向采集

有些 B2B 网站需要从一些指定网站采集求购信息，也就是询盘信息（Inquiry）。询盘信息往往需要登录到一些行业网站后抓取。首先生成每个网站的配置信息，然后根据配置信息采集。例如，根据目录页遍历求购信息。

另外，一些 B2B 网站需要抓取指定的几十个网站。首先自动查找目录页，然后提取详细页中的信息。

### 3.1 下载网页的基本方法

使用 Linux 下的 `wget` 命令递归的下载 HTTP 服务器某个目录下的所有文件。例如，递归下载所有在“ddd”目录下的文件。

```
#wget -r -np -nH -cut-dirs=3 -R index.html http://hostname/aaa/bbb/ccc/ddd/
```

使用到的选项解释如下所示。

- recursively (-r), 递归遍历。
- 不到上级目录, 例如 ccc/... (-np)。
- 不保存文件到域名文件夹(-nH)。
- 除了“ddd”文件, 忽略前 3 个文件夹“aaa”“bbb”“ccc (-cut-dirs=3) ”。
- 不包含 index.html 文件 (-R index.html)。

### 3.1.1 网卡

如果抓取的网站允许每个 IP 有 3 个连接, 那么再用一个 IP, 就可以得到 6 个连接。让爬虫使用某个网卡绑定的 IP 地址。

```
//指定本地 IP 地址 192.168.103.15
RequestConfig config = RequestConfig.custom()
    .setLocalAddress(InetAddress.getByAddress(
        new byte[] {(byte)192, (byte)168, 103, 15}))
    .build();
HttpGet httpGet = new HttpGet("http://www.lietu.com");
httpGet.setConfig(config);
CloseableHttpClient httpClient = HttpClientBuilder.create().build();
try {
    CloseableHttpResponse response = httpClient.execute(httpGet);
    try {
        //执行抓取任务
    } finally {
        response.close();
    }
} finally {
    httpClient.close();
}
```

### 3.1.2 下载网页

在 Java 语言中, java.net.URL 类能够对实际的 URL 进行建模, 通过这个类, 可以对相应的 Web 服务器发出请求并且获得相应的文档。Java.net.URL 类有一个默认的构造函数, 使用 URL 地址作为参数, 构造 URL 对象。



```
URL pageURL = new URL(path);
```

然后, 可以通过获得的 URL 对象来取得输入流, 进而像读入本地文件一样来下载网页。

```
InputStream stream = pageURL.openStream();
```

使用 **BufferedReader** 批量缓存读入, 可以指定字符集, 一般是 UTF8 或者 GBK。

```
BufferedReader reader = new BufferedReader(new InputStreamReader(
    stream, "UTF-8"));
```

然后把网页源代码读入字符串缓存。下载网页完整的代码如下。

```
String path = "http://lietu.com/";

URL pageURL = new URL(path);
InputStream stream = pageURL.openStream();
//创建网络流
BufferedReader reader = new BufferedReader(new InputStreamReader(
    stream, "UTF-8")); //字符集
String line;
//读取网页内容
StringBuilder pageBuffer = new StringBuilder(1000); //StringBuilder 是字符串缓存, 类似动态数组
while ((line = reader.readLine()) != null) { //reader.readLine 读入一行
    pageBuffer.append(line); //append 是往后面追加
}
System.out.println(pageBuffer.toString());
```

封装成一个叫作 **downloadPage** 的方法。

```
public class RetrivePage {
    public static String downloadPage(String path){
        //根据传入的路径构造 URL
        URL pageURL = new URL(path);
        //创建网络流
        BufferedReader reader = new BufferedReader(new InputStreamReader(
            pageURL.openStream()));
        String line;
        //读取网页内容
        StringBuilder pageBuffer = new StringBuilder();
        while ((line = reader.readLine()) != null) {
            pageBuffer.append(line);
        }
    }
}
```

```

        //返回网页内容
        return pageBuffer.toString();
    }
    /**
     * 测试代码
     */
    public static void main(String[] args) {
        //抓取 lietu 首页然后输出
        System.out.println(RetrivePage.downloadPage("http://www.lietu.com"));
    }
}

```

代码中的 `reader.readLine()` 有可能会抛出异常, 因为网速可能不稳定, 如果下载网页的过程中出现错误, 还需要重试。如果重试仍然出错, 下载线程可以调用 `sleep()` 方法休息片刻等待网速稳定。

Eclipse 控制台中的中文输出如果有乱码, 则检查两个地方的配置。

(1) 在 VM arguments 中添加 `-Dfile.encoding=UTF-8`。

(2) 在 Console encoding 中选择 UTF-8。

再次运行或者调试程序, 就可以在控制台中看到正确的输出了。上面的 UTF-8 是 Java 文件的编码, 如果 Java 文件编码是 GBK, 则都选择 GBK。

用 Scanner 对象下载网页的例子。

```

Scanner scanner = new Scanner(new InputStreamReader(
    pageURL.openStream(), "utf-8")); //指定编码格式 UTF-8
scanner.useDelimiter("\\z"); //可以用正则表达式分段读取网页
//读取网页内容
StringBuilder pageBuffer = new StringBuilder();
while (scanner.hasNext()){
    pageBuffer.append( scanner.next() );
}

```

很多网页的编码格式是 UTF-8, 如果读入有乱码, 则可以填入 FireFox 识别出的编码。第 4 章会专门介绍如何自动识别网页的编码。

网络中的数据是通过 TCP/IP 协议来传输的。一般使用套接字 (socket) 来对 TCP/IP 协议编程。URL 对象的 `openStream` 方法使用了 HTTP 的 GET 命令返回 Web 页的正文内容。下面是一

个直接使用套接字向 Web 服务器发送 GET 命令并输出返回结果的例子。

```
String host = "www.lietu.com"; //主机名
String file = "/index.jsp"; //网页路径
int port = 80; //端口号, 默认是 80

s = new Socket(host, port);

OutputStream out = s.getOutputStream();
PrintWriter outw = new PrintWriter(out, false);
outw.print("GET " + file + " HTTP/1.0\r\n");
outw.print("Accept: text/plain, text/html, text/*\r\n");
outw.print("\r\n");
outw.flush();//发送 GET 命令

InputStream in = s.getInputStream();
//InputStreamReader 的构造方法的第二个参数可以指定下载网页的编码格式
InputStreamReader inr = new InputStreamReader(in);
BufferedReader br = new BufferedReader(inr);
String line;
while ((line = br.readLine()) != null){
    System.out.println(line); //输出返回的网页
}
```

超代码如下所示。

```
Socket socket = new Socket(host, port);
socket.setSoTimeout(1000);
```

这个时间不是完整下载网页的时间, 而是一次 socket 读的时间。例如, 下载某个网页调用了四次套接字读取, 每次花了 9 秒, 则总的读时间是  $9 \times 4 = 36$  秒。

Web 服务器返回的结果除了网页内容, 还包括头域 (header field) 信息, 如下所示。

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=UTF-8
Content-Length: 5367
Date: Tue, 29 Jun 2010 00:32:12 GMT
Connection: close
```



网页的编码方式由 Content-Encoding 或 Content-Type 定义，它的长度由 Content-Length 或 Content-Range 定义。

URLConnection 类把头信息封装成了 HashMap 的形式，例如，key 是“Content-Length”，而 value 是“5367”。表 3-1 介绍了 HttpURLConnection 中的方法。

表 3-1 HttpURLConnection 中的方法

方 法	描 述
getHeaderField	取得头域信息
getContentType	取得网页类型
getLastModified	取得网页修改时间，如果没有则返回0
getContentEncoding	取得编码类型
getContentLength	取得网页长度

getContentType 等方法的错误在于只识别头信息中小写的字符串，需要修改成大小写不敏感的。

```
public String getHeaderField(String fieldKey) throws IOException {
    URLConnection con = url.openConnection();
    header = con.getHeaderFields(); //返回一个 HashMap

    Iterator i = getHeaderFields().keySet().iterator();
    String key = null;
    while (i.hasNext()) {
        key = (String) i.next();
        if (key == null) {
            if (fieldKey == null) {
                return (String) ((List) (getHeaderFields().get(null))).get(0);
            }
        } else {
            if (key.equalsIgnoreCase(fieldKey)) {
                return (String) ((List) (getHeaderFields().get(key))).get(0);
            }
        }
    }
    return null;
}
```

## 3.2 HTTP 基础

首先了解 HTTP 协议本身，然后看 HttpCore 如何封装使用的协议细节。

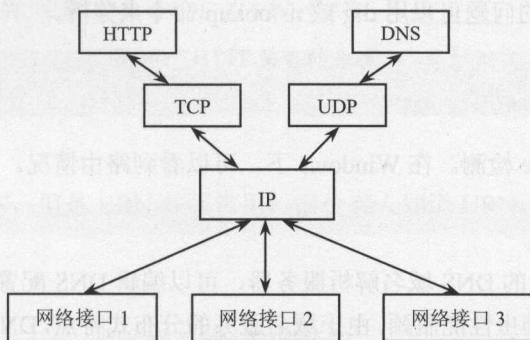


图 3-1 协议

### 3.2.1 协议

网络资源一般是 Web 服务器上的一些各种格式的文件，通过 HTTP 协议和 Web 服务器打交道，所以 Web 服务器又叫作 HTTP 服务器。HTTP 服务器存储了互联网上的数据并且根据 HTTP 客户端的请求提供数据。网络爬虫也是一种 HTTP 客户端。

在 Java 中，一旦和被采集的 Web 服务器建立连接，对网络资源的操作就好像对本地文件的操作一样简单。通常，网络中使用 URL 来标出网络资源的位置。可以直接通过 Java 中的 URL 类和存放网页的服务器建立连接，并且获得网页源代码。

Java 中的 URL 类代表一个统一的资源定位符，资源可以是简单的文件或目录，也可以是对更为复杂的对象的引用，`http://www.lietu.com/index.jsp` 是一个 URL 的例子。

通常，URL 可分成几个部分。上面的 URL 示例指示使用的协议为超文本传输协议(HTTP)，并且该信息驻留在一台名为 `www.lietu.com` 的主机上，可以调用 URL 类的 `getHost()` 方法取得 URL 地址的主机名。主机上的信息名称为 `/index.jsp`，此名称的准确含义取决于协议和主机。该信息一般存储在文件中，但可以随时生成。该 URL 的这一部分称为路径部分，可以调用 `getPath()`

方法取得路径部分。

需要通过域名服务（Domain Name Service，简称为 DNS）取得域名对应的 IP 地址。爬虫程序首先连接到一个 DNS 服务器上，由 DNS 服务器返回域名对应的 IP 地址。使用不适当的 DNS 服务器可能导致有些域名无法解析。一般情况下推荐使用 OpenDNS。

DNS 把解析到错误的域名叫作 DNS 劫持。就像当你出门问路，他给你指了错误的方向。在 Linux 下 DNS 解析的问题可以用 dig 或 nslookup 命令来分析。

```
#dig www.lietu.com
#nslookup www.lietu.com
```

或者使用 traceroute 检测。在 Windows 下，可以看到路由情况。

```
>tracert -d lietu.com
```

如果需要更换更好的 DNS 域名解析服务器，可以编辑 DNS 配置文件“/etc/resolv.conf”。DNS 解析是一个网络爬虫性能瓶颈。由于域名服务的分布式特点，DNS 可能需要多次请求转发，并在互联网上往返，需要几秒有时甚至更长的时间解析出 IP 地址。一个补救措施是引入 DNS 缓存，这样最近完成 DNS 查询的网址可能会在 DNS 缓存中找到，避免了访问互联网上的 DNS 服务器。JDK 1.6 内部有个 30 秒的 DNS 缓存，通过 sun.net.InetAddressCachePolicy.get()方法可以查看缓存时间设置。Java 中要查找一个域名的 IP 地址最方便的办法就是调用 java.net.InetAddress.getByName("www.lietu.com")。下面的代码可以得到一个网站所有的 IP 地址。

```
List<String> ipList = new ArrayList<String>();
InetAddress[] addressList = InetAddress.getAllByName(host);

for (InetAddress address : addressList) {
    ipList.add(address.getHostAddress());
}
```

URL 可选择指定一个“端口”，它是用于建立到远程主机 TCP（Transmission Control Protocol）连接的端口号。如果未指定该端口号，则使用协议默认的端口。HTTP 协议的默认端口号是 80，它可以在 URL 地址中显式指定端口号，http://www.lietu.com:80/index.jsp。

为了深入了解下载网页的原理，需要了解 HTTP 协议。HTTP 协议基于客户端/服务器模式，由客户端发起请求，服务器端应答客户端发起的请求。打个日常生活中的比方，服务器就是茶壶，客户端是茶杯，当茶杯需要水的时候，就请求茶壶分给它一些。HTTP 的客户端也叫用户代理，客户端往往是 Web 浏览器，但是在这里是网络爬虫。服务器端是网站，例如 Tomcat 或



者 Apache。客户端发起一个到服务器上指定端口（默认端口为 80）的 HTTP 请求，服务器端按指定格式返回网页或者其他网络资源，如图 3-2 所示。

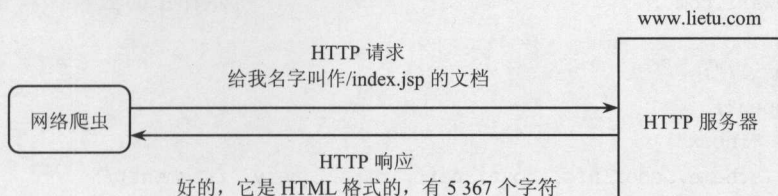


图 3-2 HTTP 协议的原理

### 3.2.2 URI

URI 包括 URL 和 URN，但是 URN 并不常用，很少有人知道 URN。URL 由 3 部分组成，如图 3-3 所示。



图 3-3 URL 的组成部分

显示 URL 地址的协议。

```
String urlString = "http://www.lietu.com/index.jsp";
URL url = new URL(urlString);
String protocol = url.getProtocol();
System.out.println(protocol); //输出 http
```

URI 的形式如下所示。

```
[scheme:]scheme-specific-part[#fragment]
```

URL 的形式如下所示。

```
http://java.sun.com/j2se/1.3/docs/guide/collections/designfaq.html#28
scheme 是 http, scheme-specific-part 是
//java.sun.com/j2se/1.3/docs/guide/collections/designfaq.html, fragment 是 28
```

URI 对象的 `isAbsolute()` 方法显示这个 URI 是否绝对路径。

```
String scheme = "http";
String userInfo = "girish";
String host = "tewari.com";
int port = 8080;
String path = "/mail/";
String query = "open";
String fragment = "inbox";
URI uri = new URI(scheme, userInfo, host, port, path, query, fragment);
System.out.println("Given uri is: " + uri);
//说明这个 URI 是否绝对路径
System.out.println("Given URI is absolute : " + uri.isAbsolute());
```

程序输出如下所示。

```
Given uri is: http://girish@tewari.com:8080/mail/?open#inbox
Given URI is absolute : true
```

经常会把 URI 写入日志，为了方便在只能显示英文的终端查看日志，URI 一般只包含一些 ASCII 码字符。对于包含中文的 URI，需要编码成一个百分号加上一个十六进制数的形式，如下所示。

```
http://lietu.com/case/价格搜索.ppt
```

编码后形式如下所示。

```
http://lietu.com/case/%E4%BB%B7%E6%A0%BC%E6%90%9C%E7%B4%A2.ppt
```

用 `URLEncoder.encode` 编码包含中文的 URL 地址，如下所示。

```
URLEncoder.encode("http://lietu.com/case/价格搜索.ppt", "UTF-8");
```

`URLEncoder.encode(term, "UTF-8")` 处理空格有问题。空格符（ASCII 码是 0x20）经过 `java.net.URLEncoder` 类 `encode` 以后，会变成+号，而不是%20，所以使用 `java.net.URI` 中的编码功能。

```
String data = new URI(null, term, null).toASCIIString();
```

用 `URLDecoder.decode` 方法还原 URL 中的编码。

```
String input = "%E6%B5%B7%E6%8A%A5%E7%BD%91";
System.out.println(URLDecoder.decode(input, "utf-8")); //用正确的编码来解码
```

客户端向服务器发送的请求头包含请求的方法、URL、协议版本，以及请求修饰符、客户信息和内容。服务器以一个状态行作为响应，相应的内容包括消息协议的版本、成功或者错误

编码，加上包含服务器信息、实体元信息以及可能的实体内容。

HTTP 请求格式如下所示。

```
<request line>
<headers>
<blank line>
[<request-body>]
```

在 HTTP 请求中，第一行是请求行，用来说明请求类型、要访问的资源以及使用的 HTTP 版本。第二行是头信息，用来说明服务器要使用的附加信息。在头信息之后是一个空行，在此之后可以添加任意的其他数据，这些附加的数据称之为主体。

HTTP 规范定义了 8 种可能的请求方法。爬虫经常用到的 3 种方法是 GET、HEAD 和 POST，说明如下所示。

- GET: 检索 URI 中标识资源的一个简单请求。例如，爬虫发送请求 GET /index.html HTTP/1.1。
- HEAD: 与 GET 方法相同，服务器只返回状态行和头标，不返回请求文档。例如，用 HEAD 请求检查网页更新时间。
- POST: 服务器接受被写入客户端输出流中的数据的请求。可以用 POST 方法来提交表单参数。

很多网页不需要接收参数，所以不接受 POST 方式发送的请求，只接受 GET 请求。

发送给 Web 服务器的请求头中可以声明可以接收的文本类型。

```
Accept: text/plain, text/html
```

这里说明了可以接收文本类型的信息，最好不要发送音频格式的数据。键/值对用冒号分隔。例如，这里的键是 Accept，值是 text/plain, text/html。

```
Referer: http://www.w3.org/hypertext/DataSources/Overview.html
```

Referer 中的值告诉服务器是从哪个页面知道这个网址的，服务器由此可以获得一些信息用于处理。有的网站会利用 Referer 防止图片盗链，所以爬虫有时候需要设置这个值。

```
Accept-Charset: GB2312,utf-8;q=0.7
```

每个语言后包括一个 q-value，表示用户对这种语言的偏好估计。默认值是 1.0，它也是最大值。



```
Keep-alive: 115
Connection: keep-alive
```

Keep-alive 是指在同一链接中发出和接收多次 HTTP 请求，单位是毫秒。

介绍完客户端向服务器的请求消息后，然后介绍服务器向客户端返回的响应消息。这种类型的消息也是由一个起始行、一个或者多个头信息、一个指示头信息结束的空行和可选的消息体组成的。

HTTP 的头信息包括通用头、请求头、响应头和实体头四个部分。每个头信息由域名、冒号(:)和域值三部分组成。域名大小写无关，域值前可以添加任何数量的空格符，头信息可以被扩展为多行，在每行开始处，使用至少一个空格或制表符，如图 3-4 所示。

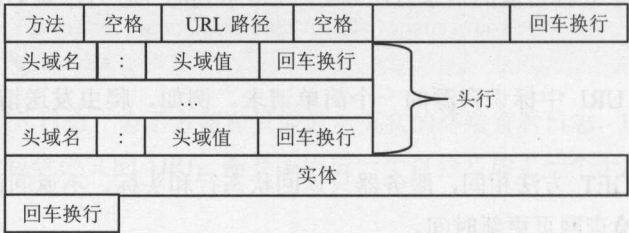


图 3-4 HTTP 请求信息格式

HTTP 的 GET 请求信息的例子如图 3-5 所示。



图 3-5 HTTP 请求信息的例子

例如，爬虫程序发出 GET 请求。

```
GET /index.html HTTP/1.1
```

服务器返回响应。

```
HTTP /1.1 200 OK
Date: Apr 11 2006 15:32:08 GMT
```

```

Server: Apache/2.0.46(win32)
Content-Length: 119
Content-Type: text/html

<HTML>
<HEAD>
<LINK REL="stylesheet" HREF="index.css">
</HEAD>
<BODY>
<IMG SRC="image/logo.png">
</BODY>
</HTML>

```

服务器返回的响应第一行就包括状态码。状态码是一个由 3 个数字组成的结果代码，爬虫可以用状态码识别 Web 服务器处理的情况。状态码的第一个数字定义响应的类别，后两个数字有分类的作用。

- 1xx：信息响应类，表示接收到请求并且继续处理。
- 2xx：处理成功响应类，表示动作被成功接收、理解和接受。
- 3xx：重定向响应类，为了完成指定的动作，必须接受进一步处理。
- 4xx：客户端错误，客户端请求包含语法错误或者是不能正确执行。
- 5xx：服务端错误，服务器不能正确执行一个正确的请求。

完整的状态码如表 3-2 所示。

表 3-2 HTTP 常用状态码

状态代码	代码描述	处理方式
200	请求成功	获得响应的内容，进行处理
201	请求完成，结果是创建了新资源。新创建资源的URI可在响应的实体中得到	爬虫中不会遇到
202	请求被接受，但处理尚未完成	阻塞等待
204	服务器端已经实现了请求，但是没有返回新的信息。如果客户是用户代理，则无须为此更新自身的文档视图	丢弃
300	该状态码不被HTTP/1.0的应用程序直接使用，只是作为3xx类型回应的默认解释。存在多个可用的被请求资源	若程序中能够处理，则进行进一步处理，如果程序中不能处理，则丢弃
301	请求到的资源都会分配一个永久的URL，这样就可以在将来通过该URL来访问此资源	重定向到分配的URL

续表

状态代码	代码描述	处理方式
302	请求到的资源在一个不同的URL处临时保存	重定向到临时的URL
304	请求的资源未更新	丢弃
400	非法请求	丢弃
401	未授权	丢弃
403	禁止	丢弃
404	没有找到	丢弃
500	服务器内部错误	丢弃
502	错误网关	丢弃
503	服务器暂时不可用	丢弃

网站 <http://www.jikexueyuan.com/robots.txt> 根本不想让来历不明的爬虫抓，可以查看 robots.txt 文本，它已经明确表示了。

```
User-agent: *  
Disallow: /**  
Disallow: /course/*.html?*
```

HTTP 请求一张图片，如果没有数据能不能做出判断？看返回的 HTTP 状态码，没有找到就返回 404 状态码。

因为 HTTP 使用的是基于文本的 TCP/IP。和一些使用二进制格式的协议不同，它直接和 Web 服务器交互，很简单。

Telnet 工具连接键盘输入到目的 TCP 端口，并且连接 TCP 端口输出返回到显示屏。Telnet 本来是用于远程终端会话。但是可以用它连接到任何 TCP 服务器，包括 HTTP 服务器。

使用 Telnet 获取 URL 地址 <http://www.lietu.com:80/index.jsp> 所在的网页。

```
# telnet www.lietu.com 80  
Trying 211.147.214.145...  
Connected to www.lietu.com (211.147.214.145).  
Escape character is '^]'.  
GET /index.jsp HTTP/1.1  
Host: www.lietu.com  
  
HTTP/1.1 200 OK  
Server: Apache-Coyote/1.1
```



```
Content-Type: text/html;charset=UTF-8
Content-Length: 5367
Date: Sat, 21 May 2011 09:59:09 GMT
```

在提交表单的时候,如果不指定方法,则默认为 GET 请求,表单中提交的数据将会附加在 URL 之后,以“?”与 URL 分开。字母、数字、字符原样发送,但空格转换为“+”,其他符号转换为“%XX”,其中 XX 为该符号以十六进制表示的 ASCII 值。GET 请求提交的数据放置在 HTTP 请求协议头中,而 POST 提交的数据则放在实体数据中。GET 方式提交的数据最多只能有 1024 字节,而 POST 则没有此限制。

```
String inputUTF8 = "%E8%BF%9B%E5%8F%A3%E5%A5%B6%E7%B2%89";
System.out.println(URLDecoder.decode(inputUTF8, "utf-8")); //输出进口奶粉

String input = "%BD%F8%BF%DA%C4%CC%B7%DB";
System.out.println(URLDecoder.decode(input, "GBK")); //输出进口奶粉
```

程序发出 POST 请求的例子。

```
try {
    //构造 POST 数据
    String data = URLEncoder.encode("key1", "UTF-8") +
        "=" + URLEncoder.encode("value1", "UTF-8");
    data += "&" + URLEncoder.encode("key2", "UTF-8") +
        "=" + URLEncoder.encode("value2", "UTF-8");

    //创建一个到 Web 服务器的套接字
    String hostname = "hostname.com";
    int port = 80;
    InetAddress addr = InetAddress.getByName(hostname);
    Socket socket = new Socket(addr, port);

    //发送头信息
    String path = "/servlet/SomeServlet";
    BufferedWriter wr =
        new BufferedWriter(new OutputStreamWriter(socket.getOutputStream(), "UTF8"));
    wr.write("POST "+path+" HTTP/1.0\r\n");
    wr.write("Content-Length: "+data.length()+"\r\n");
    wr.write("Content-Type: application/x-www-form-urlencoded\r\n");
    wr.write("\r\n");

    //发送 POST 数据
    wr.write(data);
```

```
wr.flush();

//取得响应
BufferedReader rd =
    new BufferedReader(new InputStreamReader(socket.getInputStream()));
String line;
while ((line = rd.readLine()) != null) {
    //处理读入行
}
wr.close();
rd.close();
} catch (Exception e) {
}
```

程序发出 HEAD 请求。

```
HEAD /index.jsp HTTP/1.0
```

服务器返回响应。

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=UTF-8
Content-Length: 5367
Date: Fri, 08 Apr 2011 11:08:24 GMT
Connection: close
```

使用 TCP/IP 协议来传输数据。

### 3.2.3 DNS

DNS 主要使用用户数据报协议，用户数据报协议简称 UDP（User Datagram Protocol），使用端口 53 服务请求。DNS 查询由一个单一的来源于客户端的 UDP 请求和一个服务器返回的 UDP 答复组成。当响应数据超过 512 字节时，使用 TCP。有些解析器实现对所有的查询都使用 TCP。

## 3.3 使用 HttpClient 下载网页

在实际的项目中，要处理各种复杂的网络环境。例如，需要处理 HTTP 返回的状态码，设

置 HTTP 代理, 处理 HTTPS 协议, 设置 Cookie 等工作。尽管 `java.net` 包提供了通过 HTTP 访问资源的基本功能, 但它没有提供全面的灵活性和套接字、连接池等开发爬虫需要的功能。为了便于应用程序的开发, 实际开发时常常使用开源项目 `HttpClient`。可以从 <http://hc.apache.org/downloads.cgi> 网站上下载 `HttpClient` 最新的 4.3 版本。`HttpClient` 的 JavaDoc API 说明文档可以在 `httpcomponents-core-4.3-bin.zip` 中找到。`HttpClient` 的 API 的复杂性只是反映了其问题域的复杂性。与普遍的误解相反, HTTP 是一个相当复杂的协议。`HttpClient` 4.3 以后的版本使用流式 API 让接口更简单。

`HttpClient` 用到一个基础的 jar 包——`httpcore-4.1.2.jar`、一个应用层的 jar 包——`httpclient-4.1.2.jar` 包。此外, 还使用 `commons-logging-1.1.1.jar` 记日志。

使用 `HttpClient` 获取网页内容的过程是: 首先, 创建一个 `CloseableHttpClient` 类的实例; 然后, 使用这个实例执行 HTTP 请求, 得到一个 `HttpResponse` 的实例; 最后, 通过 `HttpResponse` 的实例得到返回的二进制流, 二进制流封装在 `HttpEntity` 中。根据指定的字符集把二进制流转换成字符串, 完成下载。

`CloseableHttpClient` 类中存储了一些全局信息。创建 `CloseableHttpClient` 类的实例的代码如下所示。

```
CloseableHttpClient httpclient = HttpClientBuilder.create().build();
```

创建一个客户端, 类似于打开一个浏览器。`HttpClient` 支持所有定义在 HTTP/1.1 版本中的 HTTP 方法。对于每个方法类型都有一个特定的类, 爬虫最常用的是表示 HTTP GET 方法的 `org.apache.http.client.methods.HttpGet`, 这样是为了避免误抓登录后才能看到数据。

```
//创建一个 GET 方法, 类似于在浏览器地址栏中输入一个地址
HttpGet httpget = new HttpGet("http://www.lietu.com/");
```

使用 `CloseableHttpClient` 执行 GET 请求。

```
//类似于在浏览器地址栏中输入回车, 获得网页内容
HttpResponse response = httpclient.execute(httpget);
```

查看返回的内容, 类似于在浏览器查看网页源代码。

```
HttpEntity entity = response.getEntity();
if (entity != null) {
    //读入内容流, 并以字符串形式返回, 这里指定网页编码是 UTF-8
    System.out.println(EntityUtils.toString(entity, "utf-8")); //网页的 Meta 标签中指定了编码
```



```
EntityUtils.consume(entity);//关闭内容流
}
```

最后需要释放和 Web 服务器建立的连接。

```
httpClient.close();
```

把使用 HttpClient 下载的网页封装成一个方法。

```
public static String downloadPage(String path) throws IOException{
    //创建一个客户端，类似于打开一个浏览器
    CloseableHttpClient httpClient = HttpClientBuilder.create().build();

    //创建一个 GET 方法，类似于在浏览器地址栏中输入一个地址
    HttpGet httpget = new HttpGet(path);

    //类似于在浏览器地址栏中输入回车，获得网页内容
    HttpResponse response = httpClient.execute(httpget);

    //查看返回的内容，类似于在浏览器查看网页源代码
    HttpEntity entity = response.getEntity();
    if (entity != null) {
        //读入内容流，并以字符串形式返回，这里指定网页编码是 UTF-8
        String html = EntityUtils.toString(entity, "GBK");//网页的 Meta 标签中指定了编码
        EntityUtils.consume(entity);//关闭内容流
        return html;
    }

    return null;
}
```

调用 EntityUtils.consume 方法是为了关闭内容流，更好的方法是调用 EntityUtils.consumeQuietly(entity)方法保证完全消费了实体对象。

这个程序中，爬虫程序发出下面这样的 GET 请求得到网页。

```
GET / HTTP/1.1
```

在这个示例中，只是简单地把返回的内容打印出来，而在实际项目中，通常需要把返回的内容写入本地文件并保存，还要关闭网络连接，以免造成资源消耗。

使用 HttpGet 的例子。

```

HttpClient httpClient = new DefaultHttpClient();

HttpHost targetHost = new HttpHost("www.lietu.com");
HttpGet httpget = new HttpGet("/");
HttpResponse response = httpClient.execute(targetHost, httpget);

```

爬虫发出 GET 请求。

```
GET / HTTP/1.1
```

把下载网页封装成一个方法。

```

public static String downHtml(String url) throws IOException{ //一个简单的下载网页实现
    String content = null;

    //创建一个客户端，类似于打开一个浏览器
    DefaultHttpClient httpClient = new DefaultHttpClient();

    //创建一个 GET 方法，类似于在浏览器地址栏中输入一个地址
    HttpGet httpget = new HttpGet(url);

    //类似于在浏览器地址栏中输入回车，获得网页内容
    HttpResponse response = httpClient.execute(httpget);

    //查看返回的内容，类似于在浏览器查看网页源代码
    HttpEntity entity = response.getEntity();
    if (entity != null) {
        //读入内容流，并以字符串形式返回，这里指定网页编码是 UTF-8
        content = EntityUtils.toString(entity, "utf-8");
        EntityUtils.consume(entity); //关闭内容流
    }

    //释放连接
    httpClient.getConnectionManager().shutdown();

    return content;
}

```

从返回的请求得到字符串最简单的方法。

```

BasicResponseHandler handler = new BasicResponseHandler ();
String content = httpClient.execute(httpget, handler);

```

如果使用 `BasicResponseHandler`, 则需要自己处理碰到的异常。例如碰到 `Service Unavailable` 时, 需要自己写重试的代码。

```
BasicResponseHandler handler = new BasicResponseHandler ();
String content = null;

do{
    try{
        content = httpClient.execute(httpget, handler);
    }catch(org.apache.http.client.HttpResponseException ex){
        ex.printStackTrace();
        System.out.println("retry...");
        Thread.sleep(3000);
    }
}while(content == null);
```

可以用 `HttpHead` 得到网络资源的类型, 或者要下载的资源大小。

```
CloseableHttpClient client = HttpClientBuilder.create().build();

URI url = new URI("http://www.lietu.com/");
HttpHead method = new HttpHead(url);
HttpResponse response = client.execute(method);

Header[] s = response.getAllHeaders();

System.out.println("The header from the httpClient:");

for (int i = 0; i < s.length; i++) {
    Header hd = s[i];
    System.out.println("Header Name: " + hd.getName() + " "
        + "Header Value: " + hd.getValue());
}
```

有些网站不支持 `HEAD` 命令, 就用 `GET` 命令代替。

```
String url = "http://www.iPrima.com.cn";
HttpGet method = new HttpGet(url);
CloseableHttpClient client = CompanySite.getClient();
HttpResponse response = client.execute(method);
Header[] s = response.getHeaders("last-modified");
```



```

if(s.length==0)
    return;
String lastModified = s[0].getValue(); //!可能有多于1个的头信息
                                     //!或者根本没有
if (lastModified != null) { //防止没有得到头信息
    System.out.println("更新时间 "+lastModified);
}

```

使用 `org.apache.http.client.utils.DateUtils` 解析得到的日期字符串。

```

Date modifyDate = DateUtils.parseDate(lastModified);
System.out.println("更新时间 "+modifyDate);

```

当我们在某个网址上花太多时间去等待下载完成时，要设置超时。

```

//配置参数
int socketTimeout = 9000; //读数据超时
int connectionTimeout = 9000; //连接超时
//请求配置
RequestConfig requestConfig = RequestConfig.custom()
    .setConnectTimeout(connectionTimeout)
    .setConnectionRequestTimeout(connectionTimeout)
    .setSocketTimeout(socketTimeout).build();

CloseableHttpClient httpClient = HttpClientBuilder.create()
    .setDefaultRequestConfig(requestConfig).build();

```

`HttpClient` 的 `setSocketTimeout` 参数底层设置 `socket.setSoTimeout`。读 `timeout` 对应于一个 `socket` 读取超时，也可以设置线程超时。

```

Future<T> future = null;
future = pool.submit(new Callable<T>() {
    public T call() {
        return executeImpl(url);
    }
});

try {
    return future.get(10000, TimeUnit.SECONDS); //设置线程超时
}
catch (InterruptedException e) {

```

```

        log.warn("task interrupted", name);
    }
    catch (ExecutionException e) {
        log.error(name + " execution exception", e);
    }
    catch (TimeoutException e) {
        log.debug("future timed out", name);
    }
}

```

或者使用 `HttpAsyncClient`。

增加从头信息判断网页字符编码的过程。

```

public static String downHtml(String url){ //自动判断网页字符编码的下载网页实现
    HttpClient httpClient = new DefaultHttpClient();
    try {
        HttpGet httpget = new HttpGet(url);

        HttpResponse response = httpClient.execute(httpget);

        Pattern pattern = Pattern.compile("text/html;[\\s]*charset=(.*)");
        Header[] arr = response.getHeaders("Content-Type");
        String charset = "utf-8";
        if (arr != null || arr.length != 0) {
            String content = arr[0].getValue().toLowerCase();
            Matcher m = pattern.matcher(content);
            if (m.find()) {
                charset = m.group(1);
            }
        }

        HttpEntity entity = response.getEntity();

        if (entity != null) {
            InputStream instream = entity.getContent();
            InputStreamReader ir = new InputStreamReader(instream,
                charset);
            BufferedReader reader = new BufferedReader(ir);

            StringBuilder builder = new StringBuilder();
            char[] chars = new char[4096];
            int length = 0;

```

```

        while (0 < (length = reader.read(chars))) {
            builder.append(chars, 0, length);
        }
        return builder.toString();
    }
} catch (Exception e) {
    e.printStackTrace();
} finally {
    httpClient.getConnectionManager().shutdown();
}
return null;
}

```

这个例子是用 GET 方式来访问 Web 资源。通常，GET 请求方式把需要传递给服务器的参数作为 URL 的一部分传递给服务器。但是，HTTP 协议本身对 URL 字符串长度有所限制，因此不能传递过多的参数给服务器。为了避免这种问题，有些表单采用 POST 方法提交数据，HttpClient 包对 POST 方法也有很好的支持。例如，按城市抓取酒店信息。<http://hotels.ctrip.com/Domestic/SearchHotel.aspx> 网页中包括如下源代码信息。

```

<form name="aspnetForm" method="post" action="SearchHotel.aspx" id="aspnetForm">
    <input type="hidden" name="cityId" value="" />
    <input type="hidden" name="checkIn" value="" />
    <input type="hidden" name="checkOut" value="" />
</form>

```

POST 方法需要提交的三个参数包括：所在城市（cityId）、入住日期（checkIn）、离店日期（checkOut）。提交一个参数包括名字和值两项。NameValuePair 是一个接口，而 BasicNameValuePair 则是这个接口的实现，使用 BasicNameValuePair 封装名字/值对。例如，参数名 cityId 对应的值是 1，代码如下所示。

```
new BasicNameValuePair("cityId", "1");
```

模拟提交表单并返回结果的代码如下所示。

```

HttpClient httpClient = new DefaultHttpClient();
//使用 HttpPost 发送 POST 请求
HttpPost httpPost = new HttpPost("http://hotels.ctrip.com/Domestic/ShowHotelList.aspx");

//POST 数据
List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(3); //3 个参数
nameValuePairs.add(new BasicNameValuePair("checkIn", "2011-4-15")); //入住日期

```



```

nameValuePairs.add(new BasicNameValuePair("checkOut", "2011-4-25")); //离店日期
nameValuePairs.add(new BasicNameValuePair("cityId", "1")); //城市编码
httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

//执行 HTTP POST 请求
HttpResponse response = httpClient.execute(httpPost);

//取得内容流
HttpEntity entity = response.getEntity();
InputStream is = entity.getContent();
BufferedInputStream bis = new BufferedInputStream(is);
ByteArrayBuffer baf = new ByteArrayBuffer(20);

//按字节读入内容流到字节数组缓存
int current = 0;
while ((current = bis.read()) != -1) {
    baf.append((byte) current);
}

String text = new String(baf.toByteArray(), "gb2312"); //指定编码
System.out.println(text);

```

上面的例子说明了如何使用 POST 方法来访问 Web 资源。与 GET 方法不同, POST 方法可以提交二进制格式的数据, 因此可以传递“无限”多的参数。而 GET 方法采用把参数写在 URL 里面的方式, 由于 URL 有长度限制, 因此传递参数的长度会有限制。

HttpClient 发送 POST 的例子。

```

HttpPost httpPost = new HttpPost("https://portal.sun.com/amserver/UI/Login?" +
    "org=self_registered_users&" +
    "goto=/portal/dt&" +
    "gotoOnFail=/portal/dt?error=true");

List <NameValuePair> nvps = new ArrayList <NameValuePair>();
nvps.add(new BasicNameValuePair("IDToken1", "username"));
nvps.add(new BasicNameValuePair("IDToken2", "password"));

httpPost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.UTF_8));

response = httpClient.execute(httpPost);
entity = response.getEntity();

```

```

System.out.println("Login form get: " + response.getStatusLine());
if (entity != null) {
    entity.consumeContent();
}

System.out.println("Post logon Cookies:");
Cookies = httpClient.getCookieStore().getCookies();
if (Cookies.isEmpty()) {
    System.out.println("None");
} else {
    for (int i = 0; i < Cookies.size(); i++) {
        System.out.println("- " + Cookies.get(i).toString());
    }
}

//当不再需要HttpClient实例时, 关闭连接管理器
//以确保立即释放所有系统资源
httpClient.getConnectionManager().shutdown();

```

直接发送字符串的例子。

```

String url = "http://www.cninfo.com.cn/search/search.jsp";
//使用HttpPost 发送 POST 请求
HttpPost httpPost = new HttpPost(url);

//POST 数据
StringEntity inentity = new

StringEntity("marketType=012002&noticeType=&stockCode=%B4%FA%C2%EB%2F%BC%F2%B3%C6%2F%C6%
B4%D2%F4&keyword=&startTime=2015-05-15&endTime=2015-05-15"
);
httpPost.setEntity(inentity);

//执行 HTTP POST 请求
HttpResponse response = httpClient.execute(httpPost);

```

为了自动搜索 Google, 可以使用如下的 GET 请求。

```

HttpGet httpGet = new HttpGet
("http://www.google.com/search?hl=en&q=httpclient&btnG=Google+Search&aq=f&oq=");

```

HttpClient 提供了一系列实用的方法来简化创建和修改请求 URI。URI 可以组装编程。

```
URI uri = URIUtils.createURI("http", "www.google.com", -1, "/search",
    "q=httpclient&btnG=Google+Search&aq=f&oq=", null);
HttpGet httpget = new HttpGet(uri);
//输出 http://www.google.com/search?q=httpclient&btnG=Google+Search&aq=f&oq=
System.out.println(httpget.getURI());
```

提交的参数可以分解成 Key=Value 的形式，叫作名字/值对。HttpClient 使用专门的NameValuePair 类来表示名字/值对。通过添加参数列表来生成 Google 查询字符串的代码如下所示。

```
List<NameValuePair> qparams = new ArrayList<NameValuePair>();
qparams.add(new BasicNameValuePair("q", "httpclient")); //查询词是httpclient
qparams.add(new BasicNameValuePair("btnG", "Google Search"));
//判断搜索用户是否是第一次查询，如果用户第一次进行查询，则aq=f
qparams.add(new BasicNameValuePair("aq", "f"));
qparams.add(new BasicNameValuePair("oq", null));
URI uri = URIUtils.createURI("http", "www.google.com", -1, "/search",
    URLEncodedUtils.format(qparams, "UTF-8"), null);

HttpGet httpget = new HttpGet(uri);
//输出 http://www.google.com/search?q=httpclient&btnG=Google+Search&aq=f&oq=
System.out.println(httpget.getURI());
```

### 3.3.1 HttpCore

HttpCore 是一个开源项目。对 HTTP 协议客户端编程做了一些基本的封装。例如，格式化请求头和解析响应头。LineFormatter 用来格式化请求头信息，而实际的实现在 BasicLineFormatter 上。HttpResponseParser 解析响应头。

org.apache.http.protocol.HTTP 定义了协议中约定的一些常量。HTTP 类用变量 SP 表示空格，CR 表示回车，LF 表示换行。

请求头信息封装在一个 HttpParams。BasicHttpParams 使用一个散列表实现 HttpParams。HttpProtocolParams 包含特定的方法来设置参数，例如，设置 HTTP 协议版本号的 setVersion 方法。org.apache.http.HttpVersion 封装了所有可能的 HTTP 协议版本号。已经定义的 HTTP 协议的版本有 1.1/1.0/0.9。例如，使用 HttpProtocolParams 设置 HTTP 协议的版本为 1.1。

```
HttpParams params = new BasicHttpParams();
//设置参数到params
```



```
HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
```

设置连接参数。

```
HttpParams params = new BasicHttpParams();

//设置连接超时
HttpConnectionParams.setConnectionTimeout(params, 180 * 1000);
//设置 Socket 超时
HttpConnectionParams.setSoTimeout(params, 180 * 1000);
//设置 Socket 缓存大小
HttpConnectionParams.setSocketBufferSize(params, 8192);
```

HttpProtocolParams 有设置客户端类型的 setUserAgent 方法。

```
//把参数设置成和 IE7 相同的
HttpProtocolParams.setUserAgent(params,
    "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)");
```

HTTP 协议处理器是一个协议拦截器的集合,实现了“责任链”模式。每个协议拦截器工作在这个拦截器负责的特定方面。例如, RequestTargetHost 给请求头增加 HOST 信息, RequestUserAgent 给请求头增加 USER\_AGENT 信息。

处理请求头增加 host 域的过程。

```
//连接到 Web 服务器
DefaultHttpClientConnection conn = new DefaultHttpClientConnection();
HttpHost host = new HttpHost("www.lietu.com", 80);
Socket socket = new Socket(host.getHostName(), host.getPort());

HttpParams params = new BasicHttpParams();
//设置 HTTP 协议的版本号为 1.1
HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
conn.bind(socket, params);

//请求头信息的形成过程
BasicHttpProcessor httpProc = new BasicHttpProcessor();
//头信息增加 host 域
httpProc.addInterceptor(new RequestTargetHost());

//请求行
String targets = "/index.jsp";
BasicHttpRequest request = new BasicHttpRequest("GET", targets);
```

```

HttpRequestExecutor httpExecutor = new HttpRequestExecutor();
//上下文信息
HttpContext context = new BasicHttpContext(null);
context.setAttribute(ExecutionContext.HTTP_TARGET_HOST, host);
httpExecutor.preProcess(request, httpProc, context); //处理请求头
System.out.println(request.toString());
System.out.println(request.getHeaders("Host")[0].getValue()); //输出 www.lietu.com:80

```

为了区别用户，需要增加 Cookie 值到请求头。有些信息需要登录后才能看到，抓取这样的信息也要提供 Cookie 值。

HttpResponse 中的实体输出下载的网页内容。

```

//输出主体信息
HttpEntity entity = response.getEntity();
System.out.println(EntityUtils.toString(entity));
EntityUtils.consume(entity); //关闭内容流

```

HTTP 响应是由服务器在接收和解释请求报文之后返回发送给客户端的报文。响应报文的第一行包含了协议版本，之后是数字状态码和相关联的文本段。

```

HttpResponse response = new BasicHttpResponse(HttpVersion.HTTP_1_1,
                                                HttpStatus.SC_OK, "OK");
System.out.println(response.getProtocolVersion()); //HTTP/1.1
System.out.println(response.getStatusLine().getStatusCode()); //200
System.out.println(response.getStatusLine().getReasonPhrase()); //OK
System.out.println(response.getStatusLine().toString()); //HTTP/1.1 200 OK

```

HttpClient 是一个接口，用来代表基本的 HTTP 请求执行约定。AbstractHttpClient 实现了 org.apache.http.client.HttpClient 接口。而 DefaultHttpClient 是 HttpClient 接口的一个实现类。如果没有特别指定，DefaultHttpClient 使用 HTTP/1.1 版本，参数设置如下。

- Version: HttpVersion.HTTP\_1\_1。
- ContentCharset: HTTP.DEFAULT\_CONTENT\_CHARSET。
- NoTcpDelay: true。
- SocketBufferSize: 8192。
- UserAgent: Apache-HttpClient/release (java 1.5)。

可以自己定制 HttpClient。写一个类继承 AbstractHttpClient，叫作 CrawlerHttpClient。需要

实现 `createHttpParams` 方法。

此外, 可以使用 `AbstractHttpClient`, 也可以用 `AbstractHttpClient` 代替 `DefaultHttpClient`。

```
AbstractHttpClient httpImpl = new ContentEncodingHttpClient ();

HttpGet httpget = new HttpGet(uri);
httpget.addHeader ("User-Agent",
    "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)");
httpget.addHeader ("Accept-Encoding", "gzip,deflate");
httpImpl.execute (httpget);
```

`HttpGet` 是 `HttpRequestBase` 的子类。使用 `HttpGet` 的例子如下所示。

```
HttpClient httpClient = new DefaultHttpClient();

HttpHost targetHost = new HttpHost("www.google.cn");
HttpGet httpget = new HttpGet("/");

//查看默认 request 头部信息
System.out.println("Accept-Charset:"
    + httpget.getFirstHeader("Accept-Charset"));

httpget.setHeader("User-Agent",
    "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.1.2)");
//用逗号分隔显示可以同时接受多种编码
httpget.setHeader("Accept-Language", "zh-cn,zh;q=0.5");
//对 google.cn 来说, 无论设置 Accept-Charset 为 gbk 还是 UTF-8, 都会默认返回 gb2312
httpget.setHeader("Accept-Charset", "GB2312,utf-8;q=0.7,*;q=0.7");
HttpResponse response = httpClient.execute(targetHost, httpget);
```

`EntityUtils` 提供了一些方法来读取 `HttpEntity` 实体中的内容。

```
EntityUtils.toString(entity, "utf-8");
```

上面的代码读入实体中的内容流, 并以字符串形式返回, 这里指定实体中的内容以 UTF-8 编码。

很长的网页一次性读出来可能会出错, 按字节读又太慢, 可以使用 `Scanner` 把输入流读到 `StringBuilder` 对象中。

```
InputStream is = entity.getContent();
```



```
Scanner scanner = new Scanner(new InputStreamReader(is, "utf-8"));
scanner.useDelimiter("\\\\z"); //可以用正则表达式分段读取网页
//读取网页内容到 pageBuffer 变量
StringBuilder pageBuffer = new StringBuilder();
while (scanner.hasNext()){
    pageBuffer.append( scanner.next() );
}

System.out.println(pageBuffer.toString());
```

### 3.3.2 状态码

调用 `HttpResponse.getStatusLine()` 方法返回一个 `StatusLine` 对象，其中包含了如下所示状态码。

```
HttpGet httpget = new HttpGet("http://www.lietu.com");
HttpResponse response = httpClient.execute(httpget);
int code = response.getStatusLine().getStatusCode();
System.out.println(code); //200
```

一个 404 错误的例子。

```
String url = "http://finance.eastmoney.com/news/lewrterewrewer536663470.html";
HttpGet httpget = new HttpGet(url);
```

一个 502 错误的例子。

```
String url = "http://news.qq.com/a/20150808/00069324324348.htm";
HttpGet httpget = new HttpGet(url);
```

`HttpStatus` 接口中包含了一些常见的状态码。

```
HttpResponse response = httpClient.execute(get);
entity = response.getEntity();

if (response.getStatusLine().getStatusCode() != HttpStatus.SC_OK) {
    if (response.getStatusLine().getStatusCode() != HttpStatus.SC_NOT_FOUND) {
        logger.info("Failed: "
            + response.getStatusLine().toString()
            + ", while fetching " + toFetchURL);
    }
}
```

```
return response.getStatusLine().getStatusCode();
}
```

在 Python 语言中, 状态码中返回的 HTTP 错误可以通过 `urllib2.HTTPError` 捕获。这里写一个专门处理 404 错误的异常类。

```
public class NotFoundException extends Exception {
    public NotFoundException(String s) {
        super(s);
    }
}
```

抛出 `NotFoundException` 的例子。

```
int code = response.getStatusLine().getStatusCode();

if (code == HttpStatus.SC_NOT_FOUND) {
    httpGet.releaseConnection();
    throw new NotFoundException("url not found error:" + url);
}
```

抛出异常之前需要释放连接, 否则会报 `ConnectionPoolTimeoutException` 的错误。

软 404 错误的例子。

```
String url = "http://politics.people.com.cn/GB/1024/8888";
HttpGet httpget = new HttpGet(url);
```

### 3.3.3 创建

可以使用 `HttpClientBuilder` 或者 `HttpClients` 工具类的工厂方法创建 `HttpClient` 实例。

```
CloseableHttpClient client = HttpClients.createDefault();
```

### 3.3.4 模拟浏览器

可以调用 `HttpGet` 对象的 `setHeader` 方法来设置头信息, 代码如下所示。

```
HttpGet httpget = new HttpGet("/");
```

```
httpget.setHeader("User-Agent",  
    "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.1.2)");
```

上述代码实际发出的 GET 请求如下所示。

```
GET / HTTP/1.1  
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.1.2)
```

也可以使用 `HttpClient` 的 `setDefaultHeaders` 方法设置头信息。

```
private static List<Header> getHeads() {  
    //头信息  
    String userAgent = "Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.1.2)";  
    List<Header> headers = new ArrayList<Header>();  
    headers.add(new BasicHeader("Accept-Charset", "GB2312,utf-8;q=0.7,*;q=0.7"));  
    headers.add(new BasicHeader("Accept-Language", "zh-cn,zh;q=0.5"));  
    headers.add(new BasicHeader("User-Agent",userAgent));  
  
    return headers;  
}  
  
List<Header> headers = getHeads();  
CloseableHttpClient httpclient =  
    HttpClientBuilder.create().setDefaultHeaders(headers).build();
```

### 3.3.5 重试

按照下面的方法虽然偶尔也可以执行成功，但是有时候会出现无限等待的情况，所以一般需要限时等待。

```
HttpResponse response = httpclient.execute(httpget);  
HttpEntity entity = response.getEntity();  
InputStream instream = entity.getContent();  
content = IOUtils.toString(instream, "UTF-8");
```

做很多事情都有时间期限，例如土地使用年限 70 年，这个叫作超时时间（Timeout）。

如果要永不过期，设置成 -1 就可以。

连接请求发出去以后，如果找不到目标服务器，程序不会停留在那里，而是立即返回无法找到服务器，叫作连接超时。服务器接受了请求之后，没有在限定时间里面做出回应，叫作读



超时。

`HttpRequestRetryHandler` 接口决定执行 HTTP 请求时,碰到一个可恢复的异常后是否可以重试。`DefaultHttpRequestRetryHandler` 类实现 3 次重试,多重试 2 次的代码如下所示。

```
HttpRequestRetryHandler retryHandler =
    new StandardHttpRequestRetryHandler(5, true); //重试 5 次
CloseableHttpClient httpClient =
    HttpClientBuilder.create().setRetryHandler(retryHandler).build();
```

修改超时设置的代码。

```
//配置
int socketTimeout = 5000;
int connectionTimeout = 5000;
//求配置
RequestConfig requestConfig = RequestConfig.custom()
    .setConnectTimeout(connectionTimeout)
    .setSocketTimeout(socketTimeout)
    .build();

//创建客户端
HttpClient httpClient = HttpClientBuilder.create()
    .setDefaultRequestConfig(requestConfig).build();
```

一个使用连接管理器的例子。

```
final AbstractHttpParams httpParameters = new BasicHttpParams();
//设置连接超时时间
HttpConnectionParams.setConnectionTimeout(httpParameters, 5000);
//设置读超时时间
HttpConnectionParams.setSoTimeout(httpParameters, 5000);
ConnManagerParams.setMaxTotalConnections(httpParameters, 20);

SchemeRegistry schemeRegistry = new SchemeRegistry();
schemeRegistry.register(new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));

final ThreadSafeClientConnManager cm = new ThreadSafeClientConnManager(httpParameters,
    schemeRegistry);

client = new DefaultHttpClient(cm, httpParameters);
HttpRequestRetryHandler retryHandler = new DefaultHttpRequestRetryHandler(3, true);
```

```
client.setHttpRequestRetryHandler(retryHandler);
```

在任何情况下都重试。

```
public class AnyHttpRequestRetryHandler implements HttpRequestRetryHandler{
    @Override
    public boolean retryRequest(IOException arg0, int arg1, HttpContext arg2) {
        return true;
    }
}
```

### 3.3.6 抓取压缩的网页

有些网站页面内容返回格式为 **gzip** 压缩格式，所以在得到返回结果后要判断内容是否压缩过，如果是，则先要解压缩，然后解析内容。这样的网页返回的头信息会说明 **Content-Encoding: gzip**。

```
public class ClientGzipContentCompression {

    public final static void main(String[] args) throws Exception {
        DefaultHttpClient httpClient = new DefaultHttpClient();
        try {
            httpClient.addRequestInterceptor(new HttpRequestInterceptor() {
                public void process(
                    final HttpRequest request,
                    final HttpContext context) throws HttpException, IOException {
                    if (!request.containsHeader("Accept-Encoding")) {
                        request.addHeader("Accept-Encoding", "gzip");
                    }
                }
            });

            httpClient.addResponseInterceptor(new HttpResponseInterceptor() {
                public void process(
                    final HttpResponse response,
                    final HttpContext context) throws HttpException, IOException {
                    HttpEntity entity = response.getEntity();
                    Header ceheader = entity.getContentEncoding();
                    if (ceheader != null) {
                        HeaderElement[] codecs = ceheader.getElements();
                    }
                }
            });
        }
    }
}
```

```

        for (int i = 0; i < codecs.length; i++) {
            if (codecs[i].getName().equalsIgnoreCase("gzip")) {
                response.setEntity(
                    new GzipDecompressingEntity(response.getEntity()));
                return;
            }
        }
    }
}

});

HttpGet httpget = new HttpGet("http://www.sohu.com");

//执行 HTTP 请求
System.out.println("executing request " + httpget.getURI());
HttpResponse response = httpClient.execute(httpget);

System.out.println("-----");
System.out.println(response.getStatusLine());
System.out.println(response.getLastHeader("Content-Encoding"));
System.out.println(response.getLastHeader("Content-Length"));
System.out.println("-----");

HttpEntity entity = response.getEntity();

if (entity != null) {
    String content = EntityUtils.toString(entity, "GBK");
    System.out.println(content);
    System.out.println("-----");
    System.out.println("Uncompressed size: "+content.length());
}

} finally {
    httpClient.getConnectionManager().shutdown();
}

}

static class GzipDecompressingEntity extends HttpEntityWrapper {
    public GzipDecompressingEntity(final HttpEntity entity) {
        super(entity);
    }
}

```



```

    }

    @Override
    public InputStream getContent() throws IOException, IllegalStateException {
        //包装实体的 getContent() 决定了重复性
        InputStream wrappedin = wrappedEntity.getContent();
        return new GZIPInputStream(wrappedin);
    }

    @Override
    public long getContentLength() {
        //解压缩内容的长度未知
        return -1;
    }
}
}

```

### 3.3.7 HttpContext

为了实现简单，最初的 HTTP 被设计成无状态的协议。但是网站需要知道一个用户是否已经登录，这是需要 HTTP 请求记住的状态。**HttpContext** 表示一个 HTTP 进程的执行状态。**HTTP** 上下文的主要目的是为了促进逻辑上相关的各组件之间的信息共享。**HTTP** 上下文可以被用来存储一个消息或几个连续的消息的处理状态。如果相同的上下文在连续的几个消息之间重用，则多个逻辑相关的信息都可以参与到一个逻辑会话，例如设置 **Cookie**。

```

CookieStore CookieStore = new BasicCookieStore();
HttpContext httpContext = new BasicHttpContext();
httpContext.setAttribute(ClientContext.COOKIE_STORE, CookieStore);

```

以后可以在多个 HTTP 请求之间使用它。

```

HttpClient httpClient = new DefaultHttpClient();
//得到 method1
HttpResponse response1 = httpClient.execute(method1, httpContext);
//得到 method2
HttpResponse response2 = httpClient.execute(method2, httpContext);

```

在 HTTP 请求执行的过程中 **HttpClient** 添加属性到执行上下文，也就是执行时指定的 **HttpContext** 对象。例如，跳转的目标网页网址会放到 **ExecutionContext.HTTP\_TARGET\_HOST** 对应的值中。

```
DefaultHttpClient httpClient = new DefaultHttpClient();
HttpContext localContext = new BasicHttpContext();
HttpGet httpget = new HttpGet("http://www.google.com/");

HttpResponse response = httpClient.execute(httpget, localContext);

HttpHost target =
    (HttpHost) localContext.getAttribute(ExecutionContext.HTTP_TARGET_HOST);

System.out.println("Final target: " + target); //输出 http://www.google.com.hk
```

### 3.3.8 下载中文网站

中文网站一般只有两种编码：GBK 和 UTF8，可以写一个按指定编码下载的方法。

```
//重试下载指定编码的网页
public static String retryDown(CloseableHttpClient httpClient, String url, String encode) {
    //具体的下载实现
}
```

但这样每次调用时都要用一个额外的参数指定编码。为了能够只在下载某个指定的网站时，调用起来显得不啰嗦，每个编码用不同的子类对应。

首先创建一个下载网页的接口。

```
public interface Downer {
    public String getContent(CloseableHttpClient httpClient, String url) throws Exception;
}
```

然后创建 DownGBK 和 DownUTF8 两个子类。

```
public class DownGBK implements Downer{
    @Override
    public String getContent(CloseableHttpClient httpClient, String url) throws Exception
    {
        return HttpUtil.retryDown(httpClient, url, "gbk");
    }
}

public class DownUTF8 implements Downer{
    @Override
```

```

public String getContent(CloseableHttpClient httpclient, String url) throws Exception
{
    return HttpUtil.retryDown(httpclient, url, "utf-8");
}
}

```

使用 DownUTF8 下载 UTF-8 编码的网页。

```

String url = "http://www.lietu.com/";
CloseableHttpClient httpclient = HttpClientBuilder.create().build();
Downer downer = new DownUTF8();

String content = downer.getContent(httpclient, url);
System.out.println(content);

```

### 3.3.9 抓取需要登录的网页

很多网站的内容都只是对注册用户可见的，这种情况下就必须要求使用正确的用户名和口令登录成功后，方可浏览到想要的页面。因为 HTTP 协议是无状态的，也就是连接的有效期限只限于当前请求，请求内容结束后连接就关闭了。在这种情况下为了保存用户的登录信息必须使用到 Cookie 机制。以 JSP/Servlet 为例，当浏览器请求一个 JSP 或者是 Servlet 的页面时，应用服务器会返回一个参数，名为 jsessionid（因不同应用服务器而异），值是一个较长的唯一字符串的 Cookie，这个字符串值也就是当前访问该站点的会话标识。浏览器在每访问该站点的其他页面时都要带上 jsessionid 这样的 Cookie 信息，应用服务器根据读取的会话标识来获取对应的会话信息。

```

CookieStore CookieStore = new BasicCookieStore();
BasicClientCookie Cookie = new BasicClientCookie("JSESSIONID", getSessionId());
CookieStore.addCookie(Cookie);
client.setCookieStore(CookieStore);

```

对于需要用户登录的网站，一般在用户登录成功后会将用户资料保存在服务器的会话中，当访问到其他的页面时，应用服务器根据浏览器送上的 Cookie 中读取当前请求对应的会话标识以获得对应的会话信息，然后就可以判断用户资料是否存在于会话信息中，如果存在则允许访问页面，否则跳转到登录页面中要求用户输入账号和口令进行登录。这就是一般使用 JSP 开发网站在处理用户登录的通用方法。

对于 HTTP 的客户端来讲，如果要访问一个受保护的页面，就必须模拟浏览器所做的工作：



首先, 请求登录页面; 其次, 读取 Cookie 值; 然后, 请求登录页面并加入登录页所需的每个参数; 最后, 请求最终所需的页面。当然除第一次请求外, 其他的请求都需要附带 Cookie 信息以便服务器能判断当前请求是否已经通过验证。HttpClient (<http://hc.apache.org/httpcomponents-client-ga/>) 自动管理了 Cookie 信息, 只需要先传递登录信息, 执行登录过程, 然后直接访问想要的页面, 与访问一个普通的页面没有任何区别, 因为 HttpClient 已经帮忙发送了 Cookie 信息。下面的例子实现了这样一个访问的过程。

```
public class RenRen {
    //配置参数
    private static String userName = "邮箱地址";
    private static String password = "密码";
    private static String redirectURL =
        "http://blog.renren.com/blog/304317577/449470467"; //要抓取的网址

    //登录 URL 地址
    private static String renRenLoginURL = "http://www.renren.com/PLogin.do";

    //用于取得重定向地址
    private HttpResponse response;
    //在一个会话中用到的 httpclient 对象
    private DefaultHttpClient httpClient = new DefaultHttpClient();

    //登录到页面
    private boolean login() {
        //根据登录页面地址初始化 httpost 对象
        HttpPost httpost = new HttpPost(renRenLoginURL);
        //POST 给网站的所有参数
        List<NameValuePair> nvps = new ArrayList<NameValuePair>();
        nvps.add(new BasicNameValuePair("origURL", redirectURL));
        nvps.add(new BasicNameValuePair("domain", "renren.com"));
        nvps.add(new BasicNameValuePair("isplogin", "true"));
        nvps.add(new BasicNameValuePair("formName", ""));
        nvps.add(new BasicNameValuePair("method", ""));
        nvps.add(new BasicNameValuePair("submit", "登录"));
        nvps.add(new BasicNameValuePair("email", userName));
        nvps.add(new BasicNameValuePair("password", password));
        try {
            httpost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.UTF_8));
            response = httpClient.execute(httpost);
        } catch (Exception e) {
```

```

        e.printStackTrace();
        return false;
    } finally {
        httpost.abort();
    }
    return true;
}

//取得重定向地址
private String getRedirectLocation() {
    Header locationHeader = response.getFirstHeader("Location");
    if (locationHeader == null) {
        return null;
    }
    return locationHeader.getValue();
}

//根据重定向地址返回内容
private String getText(String redirectLocation) {
    HttpGet httpget = new HttpGet(redirectLocation);
    //创建一个响应处理器
    ResponseHandler<String> responseHandler = new BasicResponseHandler();
    String responseBody = "";
    try {
        //取得网页内容
        responseBody = httpClient.execute(httpget, responseHandler);
    } catch (Exception e) {
        e.printStackTrace();
        responseBody = null;
    } finally {
        httpget.abort();
        httpClient.getConnectionManager().shutdown();//关闭连接
    }
    return responseBody;
}

public void printText() {
    if (login()) {
        String redirectLocation = getRedirectLocation();
        if (redirectLocation != null) {
            System.out.println(getText(redirectLocation));
        }
    }
}

```

```

    }
}

public static void main(String[] args) {
    RenRen renRen = new RenRen();
    renRen.printText();
}
}

```

抓取需要 ASP.NET 表单验证的网页更加复杂, 涉及网页自动生成的一些隐藏字段。可以在 ASP.NET 的页面中的 `<form runat="server">` 控件中放置一个名叫 `VeiwState` 的隐藏域, 来定义页面的状态。

源代码如下所示。

```

<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTI0ODU5MDElOzs+ZBCF2ryjMpeVgUrY2eTj79HNl4Q=" />

.....some code

</form>

```

在配置中使用 `<pages enableEventValidation="true"/>`, 或者在页面中使用 `<%@ Page EnableEventValidation="true" %>` 将启用事件验证。出于安全目的, 此功能验证回发或回调事件的参数是否来源于最初呈现这些事件的服务器控件。此时会在页面中生成一个 `__EVENTVALIDATION` 隐藏字段。所以需要提交 `__VIEWSTATE` 和 `__EVENTVALIDATION` 两个隐藏字段。抓取网页的主要代码如下所示。

```

String url = "http://www.2552.net/Book/LC/1.aspx";
//构造HttpClient的实例
HttpClient httpClient = new DefaultHttpClient();
HttpPost httpPost = new HttpPost(url);
List<NameValuePair> nvps = new ArrayList<NameValuePair>();
nvps.add(new BasicNameValuePair("__VIEWSTATE", "省略")); //这里填实际的值
nvps.add(new BasicNameValuePair("__EVENTTARGET", "_ctl0:pager"));
nvps.add(new BasicNameValuePair("__EVENTARGUMENT", "2"));
httpPost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.UTF_8));

ResponseHandler<String> responseHandler = new BasicResponseHandler();

```



```
String responseBody = "";
try {
    responseBody = httpClient.execute(httpPost, responseHandler);
} catch (Exception e) {
    e.printStackTrace();
    responseBody = null;
} finally {
    httpPost.abort();
    httpClient.getConnectionManager().shutdown();
}
System.out.println(responseBody);
```

查询\_\_VIEWSTATE 和 \_\_EVENTVALIDATION 两个隐藏字段的值。

```
String url = "http://www.925city.cn/cityadmin/StockSearch.aspx";
HttpClient httpClient = new DefaultHttpClient();
HttpPost httpPost = new HttpPost(url);

ResponseHandler<String> responseHandler = new BasicResponseHandler();
String responseBody = "";
try {
    responseBody = httpClient.execute(httpPost, responseHandler);
} catch (Exception e) {
    e.printStackTrace();
    responseBody = null;
    return;
} finally {
    httpPost.abort();
}

//从网页上提取值
int state = responseBody.indexOf("id=\"__VIEWSTATE\"");
String viewState = responseBody.substring(state + 24);
viewState = viewState.substring(0, viewState.indexOf("\""));
int c = responseBody.indexOf("id=\"__EVENTVALIDATION\"");
String validation = responseBody.substring(c + 30);
validation = validation.substring(0, validation.indexOf("\""));

System.out.println("VIEWSTATE:"+viewState);
System.out.println("VALIDATION:"+validation);

//查询商品库存情况
```

```

httpost = new HttpPost(url);
List<NameValuePair> nvps = new ArrayList<NameValuePair>();
nvps.add(new BasicNameValuePair("__VIEWSTATE", viewState));
nvps.add(new BasicNameValuePair("__EVENTVALIDATION", validation));
nvps.add(new BasicNameValuePair("txtbn", "b11312")); //商品编号
nvps.add(new BasicNameValuePair("btnSearch", "查询"));

httpost.setEntity(new UrlEncodedFormEntity(nvps, HTTP.ASCII));

try {
    responseBody = httpClient.execute(httpost, responseHandler);
} catch (Exception e) {
    e.printStackTrace();
    responseBody = null;
} finally {
    httpost.abort();
    httpClient.getConnectionManager().shutdown();
}
System.out.println(responseBody);

```

### 3.3.10 代理

使用代理的例子。

```

DefaultHttpClient httpclient = new DefaultHttpClient();

httpclient.getCredentialsProvider().setCredentials(
    new AuthScope("202.96.110.20", 1080),
    new UsernamePasswordCredentials("username", "password"));

HttpHost targetHost = new HttpHost("http://www.lietu.com", 80, "http");
HttpHost proxy = new HttpHost("baidu.unblockwebproxysites.com", 80);

httpclient.getParams().setParameter(ConnRoutePNames.DEFAULT_PROXY, proxy);

HttpGet httpget = new HttpGet("/");

System.out.println("executing request: " + httpget.getRequestLine());
System.out.println("via proxy: " + proxy);
System.out.println("to target: " + targetHost);

```

```

HttpResponse response = httpClient.execute(targetHost, httpget);
HttpEntity entity = response.getEntity();

System.out.println(response.getStatusLine());

```

### 3.3.11 DNS 缓存

HttpClient 抓取每个 URL 时, JVM 都会自动缓存这个 URL 和对应的 IP, 并且永远缓存, 除非缓存的内容大于 JVM 的限制, 如果将来这个 URL 更换了 IP, HttpClient 会首先去 JVM 的缓存里取, 如果取到了直接根据这个 IP 去抓取。所以往往某个域名更换了 IP, 抓取结果都是 604 错误。

解决办法是在 Java 代码里添加如下所示的代码。

```
java.security.Security.setProperty("networkaddress.cache.ttl" , "0");
```

DefaultHttpClient 对象使用系统的 DNS 解析。

```

DnsResolver dnsResolver = new SystemDefaultDnsResolver();
X509HostnameVerifier hostnameVerifier = new AllowAllHostnameVerifier();
SSLContext sslcontext = SSLContexts.createSystemDefault();
RedirectStrategy redirectStrategy = new LaxRedirectStrategy();

ManagedHttpClientConnectionFactory connFactory = new ManagedHttpClientConnectionFactory(
    new DefaultHttpRequestWriterFactory(),
    new DefaultHttpResponseParserFactory());

Registry<ConnectionSocketFactory> socketFactoryRegistry = RegistryBuilder
    .<ConnectionSocketFactory> create()
    .register(
        "https",
        new SSLConnectionSocketFactory(sslcontext,
            hostnameVerifier))
    .register("http", new PlainConnectionSocketFactory()).build();

PoolingHttpClientConnectionManager manager = new PoolingHttpClientConnectionManager(
    socketFactoryRegistry, connFactory, dnsResolver);
manager.setMaxTotal(1000);
CloseableHttpClient httpClient = HttpClientBuilder.create()

```



```
.setUserAgent("Mozilla").setConnectionManager(manager)
.setRedirectStrategy(redirectStrategy).setMaxConnPerRoute(-1)
.build();
```

```
RequestConfig defaultConfig = RequestConfig.custom()
    .setCookieSpec(CookieSpecs.IGNORE_COOKIES)
    .setExpectContinueEnabled(false)
    .setStaleConnectionCheckEnabled(false)
    .setRedirectsEnabled(true)
    .setStaleConnectionCheckEnabled(false).setMaxRedirects(5)
    .build();
```

```
RequestConfig rConfig = RequestConfig.copy(defaultConfig)
    .setSocketTimeout(15000).setConnectionRequestTimeout(-1)
    .setConnectTimeout(15000).build();
```

```
HttpGet httpget = new HttpGet("http://www.sina.com.cn");
httpget.setConfig(rConfig);
```

```
HttpResponse response = httpClient.execute(httpget);
```

### 3.3.12 并行下载

多线程方式执行请求。使用连接池管理器，例如 `PoolingClientConnectionManager`。需要使用 `HttpClient 4.2` 以上的版本才有 `PoolingClientConnectionManager` 这个类。`HttpClient` 实例使用连接池管理器。

```
SchemeRegistry schemeRegistry = new SchemeRegistry();
schemeRegistry.register(
    new Scheme("http", 80, PlainSocketFactory.getSocketFactory()));

ClientConnectionManager cm = new PoolingClientConnectionManager(schemeRegistry);
HttpClient httpClient = new DefaultHttpClient(cm);
```

启动 4 个线程执行 4 个请求。

```
//要执行 GET 的 URI
String[] urisToGet = {
    "http://www.domain1.com/",
    "http://www.domain2.com/",
```

```

    "http://www.domain3.com/",
    "http://www.domain4.com/"
};

//为每个 URI 创建一个线程
GetThread[] threads = new GetThread[urisToGet.length];
for (int i = 0; i < threads.length; i++) {
    HttpGet httpget = new HttpGet(urisToGet[i]);
    threads[i] = new GetThread(httpClient, httpget);
}

//启动线程
for (int j = 0; j < threads.length; j++) {
    threads[j].start();
}

//等待子线程执行结束
for (int j = 0; j < threads.length; j++) {
    threads[j].join();
}

```

多个线程可以共用一个 `HttpClient` 实例，但是最好每个线程都有一个独立的 `HttpContext` 实例。

```

static class GetThread extends Thread {

    private final HttpClient httpClient;
    private final HttpContext context;
    private final HttpGet httpget;

    public GetThread(HttpClient httpClient, HttpGet httpget) {
        this.httpClient = httpClient; //共享的
        this.context = new BasicHttpContext(); //新建立的
        this.httpget = httpget;
    }

    @Override
    public void run() {
        try {
            HttpResponse response = this.httpClient.execute(this.httpget, this.context);
            HttpEntity entity = response.getEntity();

```

```

        if (entity != null) {
            // do something useful with the entity
        }
        //保证连接释放给管理器
        EntityUtils.consume(entity);
    } catch (Exception ex) {
        this.httpget.abort();
    }
}
}

```

PoolingClientConnectionManager 会基于它的配置分配连接。如果所有连接都已经用了，则会阻塞一个连接的请求。设置一个等待时间，如果 HttpClient 实例在给定的时间内还没能够拿到连接，就抛出 ConnectionPoolTimeoutException 异常。

对于大文件，可以每个线程下载一部分文件内容，写入到一个单独的临时文件，当所有线程都完成下载时，再将这些临时文件合并成一个。

## 3.4 下载网络资源

做生意经常需要签协议，爬虫和网站打交道，把数据抓下来，也有一些相关的协议。首先介绍和网站打交道所使用的相关协议，然后用 Java 中现成的类实现最基本的下载网页功能。为了完成一些扩展功能，再介绍使用专门的开源项目下载网页。

### 3.4.1 重定向

客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同的页面，或通过代理服务器重复该请求。

HttpClient 可以处理如下任何与重定向相关的响应代码。

- 301 永久移动，HttpStatus.SC\_MOVED\_PERMANENTLY。
- 302 临时移动，HttpStatus.SC\_MOVED\_TEMPORARILY。



- 303 参见其他, `HttpStatus.SC_SEE_OTHER`。
- 307 临时重定向, `HttpStatus.SC_TEMPORARY_REDIRECT`。

例如, `redirect.cgi` 发送 302 响应, 同时提供一个 `Location` 头信息指向 `redirect2.cgi`。

```
HTTP/1.1 302 Moved
Date: Sat, 15 May 2004 19:30:49 GMT
Server: Apache/2.0.48 (Fedora)
Location: /cgi-bin/jccook/redirect2.cgi
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
```

通过 `RedirectStrategy` 处理跳转的网址自己的 `RedirectStrategy`。

```
public class MyRedirectStrategy extends DefaultRedirectStrategy{
    @Override
    public URI getLocationURI(
        final HttpRequest request,
        final HttpResponse response,
        final HttpContext context) throws ProtocolException {
        //抓取头位置
        System.out.println(Arrays.toString(response.getHeaders("Location")));
        return super.getLocationURI(request, response, context);
    }
}
```

使用 `MyRedirectStrategy`。

```
MyRedirectStrategy myRedirectStrategy= new MyRedirectStrategy();

HttpClient instance =
    HttpClientBuilder.create().setRedirectStrategy(myRedirectStrategy).build();
HttpResponse response = instance.execute(new HttpGet("http://www.360buy.com"));
```

如果使用 `HttpClient`, 则可以通过如下代码得到跳转的网址。

```
HttpResponse response = httpclient.execute(httppost);
response.getLastHeader("Location"); //跳转的网址
response.getStatusLine().getStatusCode(); //返回的状态码
```

需要使用自己的上下文来得到当前 URL。

```
String url =

"http://www.baidu.com/link?url=tjQTdlkeyjB_MrSNGrFDWSdsIzWfvqyNsmlMFx_7kpynFOaNXRR81Sp5R7d4BZMu";

HttpGet request = new HttpGet(url);

HttpContext context = new BasicHttpContext();
ResponseHandler<String> responseHandler = new BasicResponseHandler();
String result = client.execute(request, responseHandler, context);
URI finalUrl = request.getURI();
RedirectLocations locations = (RedirectLocations)
context.getAttribute(HttpClientContext.REDIRECT_LOCATIONS);
if (locations != null) {
    finalUrl = locations.getAll().get(locations.getAll().size() - 1);
}
System.out.println(finalUrl);
```

改写出 SimpleRedirectStrategy，去掉抛出 CircularRedirectException 的代码。

```
HttpClient client = HttpClientBuilder.create().
    setRedirectStrategy(new SimpleRedirectStrategy())
    .build();

String url =
"http://www.baidu.com/link?url=dg_gFWbVjlnOcLhu3iABkPmzDObl36z8eqY5L96CNgGRALFp2mt6Ua-rwC_h8MJN9p38SLL3qXwtompp1UEoEo6aTNXU2yxwHaK6I3lT0Ly";
HttpGet request = new HttpGet(url);
```

另外有些网页通过 JavaScript 跳转实现重定向，以如下网址为例。

```
http://www.universidadperu.com/empresas/aliterm-sociedad-anonima-cerrada.php
```

查看这个网页的代码。重定向的代码就是 JavaScript。

```
<form name="InfoEmpresa" action="" id="InfoEmpresa" onsubmit="var work
= document.InfoEmpresa.buscaempresa.value;work=work.replace(/ /g,
'-');work=work.toLowerCase();window.location='http://www.universidadperu.com/empresas/bu
squeda/'
+ work; return false;">
```

如果搜索公司名“BANCO CONTINENTAL”，则用正则表达式把“BANCO CONTINENTAL”中间的空格替换成“-”，然后小写化之后跳转到如下网址。

```
http://www.universidadperu.com/empresas/busqueda/banco-continental
```

### 3.4.2 解决套接字连接限制

在 Windows 平台下，当爬虫运行多日以后，由于操作系统对于进程废弃的网络连接回收不完全，导致爬虫出错，只有重启操作系统后才能继续运行爬虫。所以要考虑使用连接池来重用网络连接。

Linux 对连接 socket 数量也有限制，可以通过系统配置增加 socket 数量的上限。当然也可以使用 socket connection pool 连接池循环利用 socket 连接。

增加短暂的端口范围，并减少 `fin_timeout`。用下面两个命令发现默认值。

```
#sysctl net.ipv4.ip_local_port_range
#sysctl net.ipv4.tcp_fin_timeout
```

短暂端口范围定义了一个主机可以从一个特定的 IP 地址创建的除 socket 之外最大的数量。`fin_timeout` 定义了这些 socket 保持在 TIME\_WAIT 状态的最小时间。

通常系统默认值如下所示。

```
net.ipv4.ip_local_port_range = 32768 61000
net.ipv4.tcp_fin_timeout = 60
```

这基本上意味着在任何时候系统不能保证多于  $(61000 - 32768) / 60 = 470$  socket。如果觉得还不够多，可以从增加 `port_range` 开始，设置范围到 15000 61000 很常见。可以通过减少 `fin_timeout` 进一步增加可获得的连接。同时做这两步，可以有 1500 个连接。

`sysctl` 是一个允许改变正在运行中的 Linux 系统的接口。它包含一些 TCP/IP 堆栈和虚拟内存系统的高级选项，这可以让有经验的管理员提高系统性能，可以使用 `sysctl` 修改系统变量。

```
#sysctl -w net.ipv4.tcp_fin_timeout=30
#sysctl -w net.ipv4.ip_local_port_range="15000 61000"
```

也可以通过编辑 `sysctl.conf` 文件来修改系统变量。`sysctl.conf` 看起来很像 `rc.conf`，它用 `variable=value` 的形式来设定值。修改配置文件，增加如下行到 `/etc/sysctl.conf` 中。

```
# increase system IP port limits
net.ipv4.ip_local_port_range = 1024 65535
```

可以使用如下的命令检查是否已经正确设置了这个值。



```
#cat /proc/sys/net/ipv4/ip_local_port_range
```

当一个新的连接请求进来的时候, 连接池管理器检查连接池中是否包含任何没用的连接, 如果有, 就返回一个。如果连接池中所有的连接都忙并且最大的连接池数量没有达到, 就创建新的连接并且增加到连接池。当连接池中在用的连接达到最大值, 所有的新连接请求进入队列, 直到一个连接可用或者连接请求超时。

HttpClient 有自己的连接池管理器。PoolingHttpClientConnectionManager 管理一个连接池, 能够为多个执行线程的连接请求提供服务, 按路由提供连接缓存。

PoolingHttpClientConnectionManager 维护最大的连接限制。每个路由不超过 2 个并行连接, 总共不超过 20 个连接, 可以使用 HTTP 参数调整连接限制。

```
PoolingHttpClientConnectionManager connPoolControl =
    new PoolingHttpClientConnectionManager();
connPoolControl.setMaxTotal(50);
```

使用连接池。

```
CloseableHttpClient httpClient = HttpClients.custom()
    .setConnectionManager(connPoolControl)
    .build();
```

### 3.4.3 下载图片

有的新闻中包含图片, 有的电商网站中的价格是图片, 介绍几种下载图片的方法。

方法一, 用 Jsoup 下载图片。

```
public static void getImg(String src,String imageFileName) throws IOException {
    Response request = Jsoup
        .connect(src)
        .referrer(src)
        .userAgent(
            "Mozilla/5.0 (Windows NT 5.1; rv:2.0b6) Gecko/20100101
Firefox/4.0b6")
        .execute();

    byte[] imgdata = request.bodyAsBytes();
    FileUtils.writeByteArrayToFile(new File(imageFileName), imgdata);
}
```

}

这里的 FileUtils 位于 commons-io-2.4.jar，可以从 <http://commons.apache.org/io/> 上下载。

方法二，用 HttpClient 下载图片。

```
/**
 *
 * @param picUrl 图片连接
 * @param imgPath 图片要保存的地址
 */
public static void downloadImg(String picUrl, String imgPath){
    CloseableHttpClient httpClient = HttpClients.createDefault();
    String name=FilenameUtils.getName(picUrl);
    try {
        HttpGet get = new HttpGet(picUrl);
        HttpResponse response = httpClient.execute(get);
        HttpEntity entity = response.getEntity();
        InputStream in = entity.getContent();

        String dir = imgPath;
        File file = new File(dir, name);
        try {
            FileOutputStream fout = new FileOutputStream(file);
            int l = -1;
            byte[] tmp = new byte[1024];
            while ((l = in.read(tmp)) != -1) {
                fout.write(tmp, 0, l);
            }
            fout.flush();
            fout.close();
        } finally {
            //关闭底层流
            in.close();
        }
    } catch (Exception e1) {
        System.out.println("下载图片出错" + picUrl);
    }
    httpClient.close();
}
```

为了减少存储空间，需要把 BMP 格式的图片转换成 jpg 格式。javax.imageio.ImageIO 类的

write 方法可以把 BufferedImage 对象保存成 jpg 格式。

```
public class ReadImage {
    //下载图片文件并转换成 jpg 格式
    public static void download(String imageUrl,String imageFileName) {
        URL url=new URL(imageUrl);

        Image src = javax.imageio.ImageIO.read(url); //构造 Image 对象
        int width = src.getWidth(null); //得到源图宽
        int height = src.getHeight(null); //得到源图长
        BufferedImage thumb = new BufferedImage(width / 1, height / 1,
            BufferedImage.TYPE_INT_RGB);
        //绘制缩小后的图
        thumb.getGraphics().drawImage(src, 0, 0, width / 1, height / 1, null);
        File file = new File(imageFileName); //输出到文件流
        ImageIO.write(thumb,"jpg", file);
    }
    public static void main(String[] args) throws Exception {
        ReadImage.download("http://www.51766.com/img/jhzdd/1167040251883.bmp",
            "D:/HH.jpg");
    }
}
```

使用 HttpClient 下载图像。

```
HttpResponse response = defaultHttpClient.execute(request);
InputStream stream = response.getEntity().getContent();
FileOutputStream fstream = new FileOutputStream("myFile.jpeg",true);
byte[] buffer = new byte[10024];
int count = -1;
while ((count = stream.read(buffer)) != -1) {
    fstream.write(buffer,0,count);
}
fstream.flush();
fstream.close();
stream.close();
```

使用 HttpClient 的 getEntity().writeTo 方法下载图像。

```
HttpGet get1 =
    new HttpGet("http://images.xiustatic.com/upload/goods20110923/11025865/110258650001/g1_400_400.jpg");
```



```
HttpResponse res = client.execute(get1);
File f = new File("D:/AB.jpg");
OutputStream ops = new FileOutputStream(f);
res.getEntity().writeTo(ops);
ops.close();
```

### 3.4.4 抓取视频

当用浏览器观看 Flash 视频时, Video Sniffer 试图找出视频的链接, 这样爬虫就可以下载这个 Flash 文件。

### 3.4.5 抓取 FTP

Commons VFS (<http://commons.apache.org/vfs/index.html>) 提供一个统一的 API 用于处理各种不同的文件系统, 包括 FTP 中的文件或者 zip 压缩包中的文件。使用 VFS 遍历 FTP 中的文件的例子。

```
FileSystemManager manager = VFS.getManager();
FileObject ftpFile = manager.resolveFile("ftp://hcl:hcl@localhost:21/loveapple");
FileObject[] children = ftpFile.getChildren();
System.out.println("Children of " + ftpFile.getName().getURI());
for (FileObject child : children) {
    String baseName = child.getName().getBaseName();
    System.out.println("文件名: " + baseName + " -- "
        + new String(baseName.getBytes("iso-8859-1"), "UTF-8"));
}
```

Commons VFS 也可以用来上传文件, 或者访问 Linux 局域网内部的其他文件。

### 3.4.6 网页更新

有人随机选用 50 万个网页做样本, 发现 23% 的网页是每天更新的, 其中 40% 域名后缀为 .com 的网页是每天更新的。网页的半衰期 (half-life) 为 10 天, 也就是说 50 万的网页在 10 天后只剩下 25 万, 再过 10 天后就只剩下 12.5 万了。此外, 网页的变化可以归纳为泊松过程 (Poisson process) 模型。万维网的变化日新月异, 如何与万维网的变化保持同步成为爬虫的又

一个主要难题。

经常有人会问：“有没有什么新消息？”这说明人的大脑是增量获取信息的，对爬虫来说也是如此。网站中的内容经常会变化，这些变化经常反映在网站首页或者目录页。为了提高采集效率，往往考虑增量采集网页。可以把这个问题看成是被采集的 Web 服务器和存储库同步的问题。更新网页内容的基本原理是：下载网页时，记录网页下载时的时间；增量采集这个网页时，判断 URL 地址对应的网页是否有更新。

网页更新过程符合泊松过程。泊松过程是指一种累计随机事件发生次数的最基本的独立增量过程。例如，随着时间增长累计某电话交换台收到的呼唤次数就构成了一个泊松过程。网页更新时间间隔符合泊松指数分布。对于不同类型的网站采用不同的更新策略，例如.com 域名的网站更新频率较高，而.gov 域名的网站更新频率较低。

对于一些不太可能会更新的网页，只抓取一遍即可。但有些网页，例如首页或者列表页更新频率较高，所以需要隔一段时间就检测这些网页是否更新。如果只是想看看网页是否有更新，可以用 HTTP 的 HEAD 命令查看网页的最后修改时间。

```
String host = "www.lietu.com"; //主机名
String file = "/index.jsp"; //网页路径
int port = 80; //端口号
Socket s = new Socket(host, port); //建立套接字对象

OutputStream out = s.getOutputStream();
PrintWriter outw = new PrintWriter(out, false);
outw.print("HEAD " + file + " HTTP/1.0\r\n"); //发送 HEAD 命令
outw.print("Accept: text/plain, text/html, text/*\r\n");
outw.print("\r\n");
outw.flush();

InputStream in = s.getInputStream(); //返回头信息
InputStreamReader inr = new InputStreamReader(in);
BufferedReader br = new BufferedReader(inr);
String line;
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
```

上面的程序在控制台打印结果。

```
HTTP/1.1 200 OK
```

```
Server: Apache-Coyote/1.1
ETag: W/"6810-1268491592000"
Last-Modified: Sat, 13 Mar 2010 14:46:32 GMT
Content-Type: text/html
Content-Length: 6810
Date: Tue, 15 Jun 2010 01:48:27 GMT
Connection: close
```

输出结果中“Last-Modified”行记录了网页的修改时间。“Date”行返回的是 Web 服务器的当前时间，可以通过 `URLConnection` 对象取得网页的修改时间，代码如下所示。

```
URL u = new URL("http://www.lietu.com");
URLConnection http = (URLConnection) u.openConnection();
http.setRequestMethod("HEAD");
System.out.println(u + " 更新时间 " + new Date(http.getLastModified()));
```

有些网页头信息没有包括更新时间，这时候可以通过判断网页长度来检测网页是否有更新。当然也可能网页更新了，但是长度没变，实际上，这种可能性非常小。

条件下载命令可以根据时间条件下载网页。再次请求已经抓取过的页面时，爬虫往 Web 服务器发送 `If-Modified-Since` 请求头，其中包含的时间是先前服务器端发过来的 `Last-Modified` 最后修改时间戳。这样让 Web 服务器端进行验证，通过这个时间戳判断爬虫上次抓过的页面是否有修改。如果有修改，则返回 HTTP 状态码 200 和新的内容。如果没有变化，则只返回 HTTP 状态码 304，告诉爬虫，页面没有变化。这样可以大大减少在网络上传输的数据，同时也减轻了被抓取的服务器的负担。下面的代码通过套接字发送 `If-Modified-Since` 头信息。

```
outw.print("GET " + file + " HTTP/1.0\r\n");
outw.print("If-Modified-Since: Thu, 01 Jul 2011 07:24:54 GMT\r\n");
```

HTTP/1.1 中还有一个 `Etag` 可以用来判断请求的文件是否被修改，可以把 `Etag` 看成网页的版本标志。`Etag` 主要为了解决 `Last-Modified` 无法解决的一些问题。

(1) 一些文件也许会周期性的更改，但是它的内容并不改变（仅仅改变了修改时间），这个时候我们并不希望客户端认为这个文件被修改了，而重新下载。

(2) 某些文件修改得非常频繁，比如在秒以内的时间内进行修改，也就说 1 秒内修改了  $N$  次，`If-Modified-Since` 能检查到的粒度是秒级的，无法判断这种修改。

(3) 不能精确地得到某些 Web 服务器文件的最后修改时间。



在第一次抓取时记录网页的 Etag。下次发起 HTTP GET 请求一个文件，同时发送一个 If-None-Match 头，这个头的内容就是我们第一次请求时 Web 服务器返回的 Etag：6810-1268491592000。如果已经修改，则返回 HTTP 状态码 200 和新的内容。如果没有修改，则 If-None-Match 为 False，返回 HTTP 状态码 304。

```
outw.print("GET " + file + " HTTP/1.0\r\n");
outw.print("If-None-Match: \"1272af65f918cb1:84f\"\r\n");
```

可以用 Range 条件下载部分网页。比如某网页的大小是 1 000 字节，爬虫请求这个网页时用 “Range: bytes=0-500”，那么 Web 服务器应该把这个网页开头的 501 个字节发回给爬虫。

发送请求信息给 Web 服务器，要求从 2 000 070 字节开始。

```
GET /down.zip HTTP/1.0
User-Agent: NetFox
Range: bytes=2000070-
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
```

仔细看一下就会发现多了一行 “Range: bytes=2000070-”。这一行的意思就是告诉服务器，down.zip 文件从 2 000 070 字节开始传，前面的字节不用传了。

服务器收到这个请求以后，返回的信息如下所示。

```
206
Content-Length=106786028
Content-Range=bytes 2000070-106786027/106786028
Date=Mon, 30 Apr 2001 12:55:20 GMT
ETag=W/"02ca57e173c11:95b"
Content-Type=application/octet-stream
Server=Microsoft-IIS/5.0
Last-Modified=Mon, 30 Apr 2001 12:55:20 GMT
```

和前面服务器返回的信息比较一下，就会发现增加了一行。

```
Content-Range=bytes 2000070-106786027/106786028
```

返回的代码改为 206 了，而不再是 200 了。回应使用 206 状态值，表示现在开始部分传输。

从断点下载的 Java 代码如下所示。

```
URL url = new URL("http://www.sjtu.edu.cn/down.zip");
```

```

URLConnection httpConnection = (URLConnection)url.openConnection();

//设置 User-Agent
httpConnection.setRequestProperty("User-Agent", "NetFox");
//设置断点续传的开始位置
httpConnection.setRequestProperty("RANGE", "bytes=2000070");
//获得输入流
InputStream input = httpConnection.getInputStream();

```

保存文件采用 IO 包中的 `RandomAccessFile` 类。假设从 2 000 070 处开始保存文件，代码如下所示。

```

RandomAccess oSavedFile = new RandomAccessFile("down.zip", "rw");
long nPos = 2000070;
//定位文件指针到 nPos 位置
oSavedFile.seek(nPos);
byte[] b = new byte[1024];
int nRead;
//从输入流中读入字节流，然后写到文件中
while((nRead=input.read(b,0,1024)) > 0) {
    oSavedFile.write(b,0,nRead);
}

```

### 3.4.7 抓取限制应对方法

有些网站为了保护自己的内容，采取各种方法限制爬虫抓取，常见的方法有 3 种。

- 一些专业的爬虫在获取网络资源时，会在 `User-Agent` 栏中表明自己的身份。这些网站从 HTTP 请求的头信息区分浏览器和爬虫访问，只有头信息和浏览器一样才正常返回结果。
- 当来自同一个 IP 的访问次数达到一定限制值后封 IP，也就是把这个 IP 列入黑名单，超过一段时间后再解封。
- 采用 JavaScript 动态显示内容。

有些网站中的网页在浏览器中可以正常访问，但却不能用程序正常下载，这时候需要模仿浏览器下载网页。可以用 Firebug 查看 Firefox 浏览器发送的头信息。可以从 <https://www.mozilla.org/en-US/firefox/all/> 下载可以独立安装的 Firefox 浏览器。

可以发送和浏览器类似的头信息来简单地模仿浏览器访问。使用 `URLConnection` 下载

网页时，设置请求头信息。

```
URLConnection urlConnection = (URLConnection) pageURL.openConnection();
//设置和浏览器相同的头信息
urlConnection.setRequestProperty("User-Agent", "MSIE 6.0");
urlConnection.setRequestProperty("Host", pageURL.getHost());
urlConnection.setRequestProperty("Accept-Language", "ru");
urlConnection.setRequestProperty("Accept-Charset", "gb2312,utf-8;q=0.7,*;q=0.7n");
urlConnection.setRequestProperty("content-type", "text/html");
```

使用 `HttpClient` 下载网页时，例子代码如下所示。

```
DefaultHttpClient httpClient = new DefaultHttpClient();
BasicHttpParams hp = new BasicHttpParams();
hp.setParameter("http.useragent", "Mozilla/1.0 (compatible; linux 2015 plus; yep)"); // you
oughta change this into your own UA string....
httpClient.setParams(hp);

try {
    HttpGet httpget = new HttpGet("http://www.lietu.com/");
    HttpContext HTTP_CONTEXT = new BasicHttpContext();
    HttpResponse response = httpClient.execute(httpget, HTTP_CONTEXT);

    HttpEntity entity = response.getEntity();
    if (entity != null) {
        System.out.println(EntityUtils.toString(entity, "utf-8"));
        EntityUtils.consume(entity);
    }
} finally {
    httpClient.getConnectionManager().shutdown();
}
```

有些登录后才能看到的信息可以人工登录后，手动在程序中设置动态的 `Cookie` 值。

```
//假设 Cookie 值在 CookieValue 中
uc.setRequestProperty("Cookie", CookieValue);
```

有些网站对于同一个 IP 在一段时间内的访问次数有限制。可以使用 `Socket` 代理来更改请求的 IP。这时可以通过大量不同的 `Socket` 代理循环访问网站。

有些网站专门发布免费的代理列表（HTTP free proxy list），可以从这些网站抓到免费代理的 IP 地址以及端口号，然后建立有效代理列表，`proxyIP.txt` 格式如下所示。



```

219.93.178.162:3128
222.135.79.253:8080
203.160.1.38:554
132.239.17.225:3124
169.229.50.5:3124
203.160.1.146:554
203.160.1.49:554

```

建立 ProxyDB 类，循环使用这些 Socket 代理。

```

int pos = proxyIpList.get(count).toString().indexOf(":");
if (pos > 0) {
    String port = proxyIpList.get(count).toString().substring(pos+1);
    proxyAddr = proxyIpList.get(count).toString().substring(0,pos);
    proxyPort = Integer.parseInt(port);
    SocketAddress socketaddress = new InetSocketAddress(proxyAddr,proxyPort);
    proxy = new Proxy(Proxy.Type.HTTP,socketaddress);
}

```

最后通过下载网页 URL 的 openConnection 方法使用它。

```
url.openConnection(ProxyDB.getProxy());
```

通过 Modem 方式上网的计算机，每次上网所分配到的 IP 地址都不相同，这就是动态 IP 地址。因为 IP 地址资源很宝贵，大部分用户都是通过动态 IP 地址上网的。所谓动态就是指，当你每一次上网时，电信或网通会随机给你分配一个 IP 地址。重新启动上网的 Modem 就可以更换 IP 地址，使用新的 IP 地址继续抓取信息。使用浏览器 Firefox 下的插件 Firebug 分析出单击“重启路由器”时，浏览器向路由器发送的请求内容如下所示。

```

GET /userRpm/SysRebootRpm.htm?Reboot=%D6%D8%C6%F4%C2%B7%D3%C9%C6%F7 HTTP/1.1
Host:192.168.1.1
Authorization:Basic YWRtaW46YWRtaW4=

```

其中，在 Authorization 请求头的内容中，“Basic”表示“Basic authorization 验证”；“YWRtaW46YWRtaW4=”是使用 Base64 编码后的用户名和密码，解密后是“admin:admin”。

同样，也可以用程序实现以上的 HTTP 请求。

```

//Modem的用户名和密码
String data = "admin:admin";
String authorization = Base64.encode(data.getBytes());

```

```
String host = "192.168.1.1"; //Modem 的 IP 地址
String file =
"/userRpm/SysRebootRpm.htm?Reboot=%D6%D8%C6%F4%C2%B7%D3%C9%C6%F7"; //网页路径
int port = 80; //端口号

Socket s = new Socket(host, port);

OutputStream out = s.getOutputStream();
PrintWriter outw = new PrintWriter(out, false);
outw.print("GET " + file + " HTTP/1.1\r\n");
outw.print("Authorization:Basic "+authorization+"\r\n");
outw.print("\r\n");
outw.flush();
```

如果只有一台机器上网可利用 Windows 自带的 Rasdial 命令实现 ADSL 自动拨号。在程序中调用 rasdial 命令重新拨号，达到更换 IP 地址的目的。

```
public class ConnectAdslNet {
    static Logger logger = Logger.getLogger(ConnectAdslNet.class);

    /**
     * 执行 CMD 命令并返回 String 字符串，Windows 操作系统需要是 GBK 编码
     */
    public static String executeCmd(String strCmd) throws Exception {
        Process p = Runtime.getRuntime().exec("cmd /c " + strCmd);
        StringBuilder sbCmd = new StringBuilder();

        //注意编码 GBK
        BufferedReader br = new BufferedReader(new InputStreamReader(p
            .getInputStream(), "GBK"));
        String line;
        while ((line = br.readLine()) != null) {
            sbCmd.append(line + "\n");
        }
        return sbCmd.toString();
    }

    /**
     * 连接 ADSL
     */
    public static boolean connectAdsl(String adslTitle, String adslName, String adslPass)
        throws Exception {
```

```

        System.out.println("正在建立连接.");
        String adslCmd = "rasdial " + adslTitle + " " + adslName + " "
            + adslPass;
        String tempCmd = executeCmd(adslCmd);

        //判断是否连接成功
        if (tempCmd.indexOf("已连接") > 0) {
            System.out.println("已成功建立连接.");
            return true;
        } else {
            System.out.println(tempCmd);
            System.out.println("建立连接失败");
            return false;
        }
    }

    /**
     * 断开 ADSL
     */
    public static boolean disconnectAdsl(String adslTitle) throws Exception {
        String disconnectAdsl = "rasdial " + adslTitle + " /disconnect";
        String result = executeCmd(disconnectAdsl);

        if (result.indexOf("没有连接") != -1) {
            System.out.println(adslTitle + "连接不存在!");
            return false;
        } else {
            System.out.println("连接已断开");
            return true;
        }
    }

    /**
     * adsl 重新拨号, 支持失败不断重拨
     * @param args
     * @throws Exception
     */
    public static boolean reconnectAdsl(String adslTitle, String adslName, String adslPass) {
        boolean bAdsl = false;
        try {
            disconnectAdsl(adslTitle);

```



```

Thread.sleep(3000);
bAdsl = connectAdsl(adslTitle,adslName,adslPass);
Thread.sleep(3000);
int i = 0;
while (!bAdsl){
    disconnectAdsl(adslTitle);
    Thread.sleep(3000);
    bAdsl = connectAdsl(adslTitle,adslName,adslPass);
    Thread.sleep(3000);
    if(i>5){
        break;
    }
}
}catch(Exception e){
    logger.error("ADSL 拨号异常: ", e);
}

return bAdsl;
}

public static void main(String[] args) throws Exception {
    reconnectAdsl("宽带","adsl 账号","密码");
}
}

```

为了找出抓取中的问题，可以采用网络工具 Wireshark 追踪。

### 3.4.8 URL 地址提取

在 Windows 的控制台窗口中，可以根据当前路径的相对路径转移到一个路径，例如转移到当前路径的上级路径。在 HTML 网页中也经常使用相对 URL。

绝对 URL 就是不依赖其他的 URL 路径，例如，“http://www.lietu.com/index.jsp”。在一定的上下文环境下可以使用相对 URL。网页中的 URL 地址可能是相对地址，例如“./index.html”。可以在<A>和<img>标签中使用相对 URL。

```

```

可以根据所在网页的绝对 URL 地址把相对地址转换成绝对地址。为了灵活引用网站内部资

源, 相对路径在网页中很常见。爬虫为了后续处理方便, 需要把相对地址转化为绝对地址。URL 对象的 `toString` 方法返回的是绝对地址, 因此为了把相对地址转化成绝对地址, 只需要生成相对地址对应的 URL 对象即可。`new URL(fromUrl, url)` 方法可以实现从相对地址 URL 和来源 URL 对象 `fromUrl` 生成新的 URL 对象, 测试功能。

```
URL fromUrl = new URL("http://www.lietu.com/news/");
String url = "../index.html"; // 网页中的相对超链接地址
String newUrl = (new URL(fromUrl, url)).toString(); // 转成字符串类型
System.out.println(newUrl); // 打印 http://www.lietu.com/index.html
```

封装成一个方法。

```
public static String parseRealURL(String urlSource, String url)
    throws MalformedURLException {
    URL from = new URL(urlSource);
    return (new URL(from, url)).toString();
}
```

有时候需要检测网页源码中的 `base` 标签, 作为相对路径基于的 URL 地址。

```
URI a = new URI("http://www.foo.com");
URI b = new URI("bar.html");
URI c = a.resolve(b);
c.toString()    -> http://www.foo.combar.html
```

有些相对地址的解析不正确, 例如, 源地址是: `http://www.bradfordexchange.com/mcategory/apparel-and-accessories_9750/womens-jackets.html`

相对地址是: `mcategory/apparel-and-accessories_9750/womens-jackets_pg2.html`

采用如下的代码修正这个错误。

```
char firstChar = s.charAt(0);
if (firstChar >= 'a' && firstChar <= 'z' || firstChar >= 'A'
    && firstChar <= 'Z') {
    if (!s.startsWith("https:") && !s.startsWith("http:")) {
        s = "/" + s;
        URI newUri = baseURI.resolve(s);
        return newUri;
    }
}
```

HttpClient 的类 `org.apache.http.client.utils.URIUtils` 中有一个 `resolve` 方法可以用来根据相对 URL 地址生成绝对 URL 地址。`resolve` 方法接收两个值：前面一个参数是绝对 URL，后面一个参数是相对 URL，例如解析如下代码。

```
URI baseURI = URI.create("http://a/b/c/d;p?q");
System.out.println(URIUtils.resolve(this.baseURI, "/g").toString());
```

URL 地址中的“#”指向内部锚点，对于爬虫没有意义。

```
int index = href.indexOf('#'); //忽略锚点
if (index != -1) {
    href = href.substring(0, index);
}
```

使用 NekoHTML 从网页的 DOM 树提取网址的代码如下所示。

```
HashSet<String> links = new HashSet<String>();

NodeList nodeList = document.getElementsByTagName("A"); //取得 a 标签的节点列表
for (int i = 0; i < nodeList.getLength(); ++i) {
    Node n = nodeList.item(i);
    Node linkNode = n.getAttributes().getNamedItem("href");
    if(linkNode==null)
        continue;
    String href = linkNode.getNodeValue(); //取得链接值
    if("").equals(href))
        continue;

    //跳过链接，只是网页锚，忽略锚
    if (href.charAt(0) == '#') {
        continue;
    }

    //跳过mailto 链接，忽略mailto
    if (href.indexOf("mailto:") != -1) {
        continue;
    }

    //忽略 JavaScript 链接
    if (href.toLowerCase().indexOf("javascript") != -1) {
```



```

        continue;
    }

    if (href.indexOf(":/") == -1) {
        //处理绝对 URL
        if (href.charAt(0) == '/') {
            href = pageUrl.getProtocol()+(":/") + pageUrl.getHost() + href;
            //处理相对 URL
        } else {
            String file = pageUrl.getFile();
            if (file.indexOf('/') == -1) {
                href = pageUrl.getProtocol()+(":/") + pageUrl.getHost() + "/"
                    + href;
            } else {
                String path = file.substring(0, file
                    .lastIndexOf('/') + 1);
                href = pageUrl.getProtocol()+(":/") + pageUrl.getHost() + path
                    + href;
            }
        }
    }

    int index = href.indexOf('#'); //忽略锚点
    if (index != -1) {
        href = href.substring(0, index);
    }

    links.add(href);
}

```

有些网址是通过提交表单，或者通过 JavaScript 来跳转的，可以参考 HtmlUnit (<http://htmlunit.sourceforge.net/>) 中的相关实现来获取。

### 3.4.9 解析 URL 地址

URL 地址中可能有多个参数，例如，<http://lietu.com/mypage.html?value1=x&value2=y&value3=z>，URL 地址分解成两部分：“?” 前面的基本部分，和 “?” 后面的参数部分。

```
String completeURL = "http://lietu.com/mypage.html?value1=x&value2=y&value3=z";
```

```

int i = completeURL.indexOf("?");
if (i > -1) {
    String searchURL = completeURL
        .substring(completeURL.indexOf("?") + 1); //取得网页的参数部分

    StringTokenizer st = new StringTokenizer(searchURL, "&"); //按&分开参数部分
    while (st.hasMoreTokens()) {
        String searchValue = st.nextToken();
        System.out.println("value :" + searchValue); //并没有实际的用处
    }
    String baseURL = completeURL.substring(0, completeURL.indexOf("?"));
}

```

把参数解析成键/值对，放入 HashMap。

```

HashMap<String,String> searchparms = new HashMap<String,String>();
StringTokenizer st1 = new StringTokenizer(search, "&"); //首先用&分割出键/值对
while (st1.hasMoreTokens()) {
    StringTokenizer st2 = new StringTokenizer(st1.nextToken(), "="); //然后用=分割
    String key = st2.nextToken();
    String value = "";
    if (st2.hasMoreTokens()) {
        value = st2.nextToken();
    }
    searchparms.put(key, value);
}

```

### 3.4.10 归一化

判断一个 URL 是否已经抓取过。首先要把这个 URL 唯一化，也就是 URL 地址归一化。例如，一个 URL 中包含大量无效参数而实际是同一个页面，这将视为同一个 URL 来对待。

删除默认端口号。例如，把 `lietu.com:80` 转换成为 `lietu.com`。

### 3.4.11 增量采集

使用 `DocIDServer` 类记住哪些 URL 已经访问过，实现增量采集。其中，`DocIDServer.getNewDocID(url)`方法用于记住一个已经访问过的 URL，`DocIDServer.isSeenBefore(url)`方法用于

判断一个 URL 是否已经访问过。

```
boolean resumable = true;
EnvironmentConfig envConfig = new EnvironmentConfig();
envConfig.setAllowCreate(true);
envConfig.setTransactional(resumable);
envConfig.setLocking(resumable);

File envHome = new File("e:/frontier");
if (!envHome.exists()) {
    if (!envHome.mkdir()) {
        throw new Exception("Couldn't create this folder: "
            + envHome.getAbsolutePath());
    }
}

Environment env = new Environment(envHome, envConfig);
DocIDServer docIdServer = new DocIDServer(env);
String url = "lietu.com";
boolean seenBefore = docIdServer.isSeenBefore(url);
System.out.println("看见过吗: " + seenBefore);
```

### 3.4.12 iframe

提取出 iframe 中的 src 地址，然后请求这个地址内容。

```
Element iframe = doc.select("iframe").first();
String iframeSrc = iframe.attr("src");

if(iframeSrc != null){
    iframeContentDoc = Jsoup.connect(iframeSrc).get();
}
```

集成相对路径转绝对路径的完整例子。

```
String url = "http://info.tangshan.gov.cn/dept.jsp?deptid=294";
Document doc = Jsoup.connect(url).get();
Element iframe = doc.select("iframe").first();
String iframeSrc = iframe.attr("src");

if(iframeSrc != null){
    System.out.println("相对路径:"+iframeSrc);
}
```



```
String newLink = UrlResolver.resolveUrl (url, iframeSrc);
System.out.println("绝对路径:"+newLink);
}
```

### 3.4.13 抓取 JavaScript 动态页面

很多网页中包含 JavaScript 代码。例如，一些网页链接嵌入 JavaScript 代码中，因此爬虫最好能够识别 JavaScript，否则就无法遍历这样的网站了。

动态链接有两种写法。

```
"<a href="#" onclick="myJsFunc();">Run JavaScript Code</a>"
"<a href="javascript:void(0)" onclick="myJsFunc();">Run JavaScript Code</a>"
```

使用 HtmlUnit 收集动态链接地址。

```
HtmlPage page = webClient.getPage(url);
List<HtmlAnchor> anchors = page.getAnchors(); //得到一个网页中的所有锚点对象

List<HtmlAnchor> jsAnchors = new ArrayList<HtmlAnchor>(); //记录动态锚点对象的列表
for (HtmlAnchor anchor : anchors) {
    String href = anchor.getAttribute("href");
    if (href.startsWith("javascript:")) {
        jsAnchors.add(anchor);
    } else if ("#" .equals(href)) {
        if (anchor.hasAttribute("onclick")) {
            jsAnchors.add(anchor);
        }
    }
}
```

通过 HtmlAnchor 对象得到实际链接网址。

```
public static String getLink(HtmlAnchor jsAnchor) throws IOException {
    final HtmlPage newPage = jsAnchor.click();
    String jsURL = newPage.executeJavaScript("document.location")
        .getJavaScriptResult().toString(); //通过执行 JavaScript 得到网址
    return jsURL;
}
```

或者采用更简单的写法。

```
HtmlPage newPage = jsAnchor.click();
URL jsURL = newPage.getUrl();
```

Mozilla 发布了纯 Java 语言编写的 JavaScript 脚本解释引擎 Rhino (<http://www.mozilla.org/rhino/>)。Rhion 包含所有 JavaScript 1.7 的特征。Rhion 附带一个可以运行 JavaScript 脚本的 JavaScript 外壳。用一个 JavaScript 编译器把 JavaScript 源文件翻译成 Java 的 class 文件,相当于用 Java 虚拟机执行 JavaScript 代码。

有一个 JavaScript 调试器用于 Rhino 执行脚本。Rhino 可以实现从 Java 对象到动态页面脚本片段常用语言——JavaScript 对象的直接映射,这有利于简化脚本解析环境的构建工作,减少脚本解释引擎与脚本片段在实现语言方面的差异。同时,在脚本解析的过程中,Rhino 对于 JavaScript 对象的操作结果,可以通过访问已经在本地创建的、与其一一对应的 Java 对象直接获得,示例代码如下所示。

```
ScriptEngineManager factory = new ScriptEngineManager();
ScriptEngine engine = factory.getEngineByName("JavaScript");

String script = "var total = 2;";
engine.eval(script); // 加载 JavaScript
Double total = (Double)engine.get("total");
System.out.println(total);
```

数组的处理。

```
String script = "var szzbAffiches=[[\"002168\", \"finalpage/2015-03-28/1200755819.PDF\", \"深圳惠程: 长春高琦聚酰亚胺材料有限公司审计报告\", \"PDF\", \"2601\", \"2015-03-28\", \"2015-03-28 00:00\"]];";

engine.eval(script+"var JavaArray = Java.to(szzbAffiches, \"java.lang.String[][]\");"); // 加载 JavaScript
java.lang.String[][] arr=(String[][] engine.get("JavaArray"));
System.out.println(arr[0][1]);
```

Rhino 的主要功能是脚本执行时的运行环境管理。运行环境是指用来保存所要执行的脚本中的变量、对象和执行上下文的空间。运行环境中的变量和对象由运行环境内所有的执行上下文共享,即一个执行上下文创建的变量或对象,其他上下文也可以访问,运行环境负责处理变量或对象访问时的同步和互斥问题。运行环境和执行上下文是执行脚本语句的场所,因此在应用程序中应首先调用 Rhino 提供的 API 建立一个运行环境和若干个执行上下文,然后调用相应的 API 建立脚本语言的内置对象。

HTML DOM 中只有 Window 和 Document 对象的方法参数中含有超链接网络地址信息和页面主体内容。因此在进行 HTML DOM 对象本地创建时,将其余对象的属性和方法简单地设置为空 (NULL)。

在 Window 和 Document 对象的方法参数中,与超链接网络地址、页面主体内容相关的函数可以分为两类:第一类,以 Window 对象的 open 方法为代表,open 方法的参数是动态页面中的超链接网络地址,参数类型是 JavaScript 语言内置 String 类型。在引擎外创建该类方法时,声明该方法的行为是把参数,即超链接网络地址,送入信息采集环节的待获取 URL 队列中;第二类,以 Document 对象的 write 方法为代表,write 方法的参数(同为 JavaScript 语言内置 String 类型)是一段表达脚本片段最终在浏览器中呈现内容的静态网页源文件。类似于常见的静态网页,在作为 write 方法参数的网页源文件中,超链接网络地址和页面主体内容被分别以 URL 和文本信息方式直接嵌入 HTML 标记中。在引擎外创建该类方法时,声明该方法的行为是把参数,即静态网页源文件,写入位于本地特定的文件中。

由于 Rhino 能够自动在 Java 对象和 JavaScript 对象之间根据“对象名称一致性”的原则实现一一对应。因此,当 Rhino 在执行脚本片段中的“Window.open()”与“Document.write()”时,实际上是分别调用在 Java 语言作用域中定义的与这两个方法同名的 Java 函数,执行函数体中关于函数行为的描述。

在完成脚本片段提取和 HTML DOM 本地创建后,就可以调用 Rhino 提取 JavaScript 动态页面中的超链接网络地址及页面主体内容。当遇到脚本片段中的 HTML DOM 时,Rhino 根据引擎外创建的同名函数体中的行为描述执行相应动作。

提取这个跳转链接可以采用 HtmlUnit (<http://htmlunit.sourceforge.net/>) 实现。HtmlUnit 底层也是调用了 Rhino,但是绕过了 Rhino 的一些错误。htmlunit-core-js-2.14.jar 代码位于 <https://github.com/HtmlUnit/htmlunit-rhino-fork> 中。

使用 HtmlUnit 抓取淘宝的例子。

```
String[] urls = new String[]{
    "http://item.taobao.com/item.htm?id=12829488341",
    "http://item.taobao.com/item.htm?id=12245778956",
    "http://item.taobao.com/item.htm?id=5226775514",
    "http://item.taobao.com/item.htm?id=6645050729",
    "http://item.taobao.com/item.htm?id=3374461221",
    "http://item.taobao.com/item.htm?id=12386999985",
    "http://item.taobao.com/item.htm?id=9653757981",
    "http://item.taobao.com/item.htm?id=3395950822",
```



```

        "http://item.taobao.com/item.htm?id=9843881848"
    };
    WebClient webClient = new WebClient();
    webClient.setJavaScriptEnabled(false);
    for (int i = 0; i < urls.length; i++) {
        HtmlPage page = webClient.getPage(urls[i]);
        HTMLElement element = page.getFirstByXPath("//*[@id=\"J_StrPrice\"]");
        System.out.println(element.asText());
    }
}

```

HtmlUnit 对于 JavaScript 框架 jQuery 的支持不太好。

HtmlUnit 调用了 CSS 解析器 cssparser-0.9.13.jar。gngr 是一个纯 Java 实现的浏览器。

另外一种方式是用 Selenium (<http://seleniumhq.org>) 抓取 JavaScript 动态信息。PhantomJs 是一个无 UI 界面的浏览器运行环境接口系统。关于 Selenium 和 PhantomJs 有专门的介绍。

Watir (<http://watir.com/>) 是一个控制浏览器行为的类似项目。虽然可以利用 jrex (<http://freecode.com/projects/jrex>) 提取渲染后的 html 文件，但推荐用 C# 来实现。

如果只需要抓一个 AJAX 网站，可以手写模拟浏览器请求服务器端数据并提取信息的特定抓取代码。服务器端返回的数据往往是 JSON 格式的，需要解析它可以使用 GSON (<https://code.google.com/p/google-gson/>) 解析出想要的信息。

例如，抓取金山词霸中的词表。

```

[{"key":"toddler","value":1177}, {"key":"toddlers","value":133}, {"key":"toddlerhood","value":31}, {"key":"toddler age","value":7}]

```

完整的流程如下所示。

```

String word = "toddler";
String url = "http://www.iciba.com/ajax_sugg.php?key=" +
    URLEncoder.encode(word, "UTF-8");

//用 HttpClient 下载
String jsonContent = getJson(url);

//用 GSON 解析
Type type = new TypeToken<List<Map<String, String>>>() {}.getType();
Gson gson = new Gson();
List<Map<String, String>> result = gson.fromJson(jsonContent, type);

```

### 3.4.14 抓取即时信息

信息的即时性往往很重要。例如，刚出现的经济信息可能在几秒以后就会影响相关股价。

为了加快获取信息的速度，可以先只抓取标题及 URL 地址。对于首页，通过 HTTP 协议的 HEAD 命令判断页面是否已经更新。在不同的时间段，用不同的频率来检查网页更新。

很多行业网站的首页是按板块组织的，可以按板块提取 URL 地址。有的板块全部是新的，也就是说，新的信息从这个板块溢出了。对于从板块溢出的地址，再从该版块对应的索引页补全信息。查找一个板块对应的索引页的方法是：查找该板块的“更多”链接对应的 URL 地址。

为了节省带宽，抓取到某个页面时，如果已经没有新的信息，可以不再继续往下检查这个页面的后续网页是否有更新。许多 Web 服务器具有发送压缩数据的能力，这可以将网络线路上传输的大量数据消减 60% 以上。

假设一个网站的首页有  $90K \times 8 \text{ 位} = 720K \text{ 位}$ ，每秒需要同步 200 个网站，则需要大约 140M 带宽。

### 3.4.15 抓取暗网

没有入口可以用来遍历所有信息的网站对爬虫来说叫作暗网。例如，从词典网站抓取例句，需要准备好要查询的词才能抓到有用的信息。

```
String url = "http://cn.bing.com/dict/search?q=hello"; //q 后面是查询词
```

互联网就像是深不可测的大海，很多有用的信息隐藏在互联网中。如何发现新网站包含感兴趣的信息？提交查询词到通用搜索引擎。

这样的爬虫叫作深网爬虫或者暗网爬虫。

暗网的表现形式一般是：前台是一个表单来获取，提交后返回一个列表形式的搜索结果页，它们是由暗网后台数据库动态产生的。因为返回的结果往往来源于同一个模板，所以可以用同一个 DOM 解析树处理不同的页面。

如果用单线程抓某个固定的网站，下载不同的页面时，可以共享同一个网络链接，避免重复打开和关闭链接多花时间。

搜索引擎本身也可以看作是一个暗网。搜索结果页包含了指向详细内容页的链接。搜索引擎中的内容有挖掘价值。例如，有一个乡镇词表，其中“郭河”是一个小地名，但不知道它是一个镇还是一个村。把这个词提交给搜索引擎，搜索引擎返回的结果中可能包含了“郭河镇”这样的结果，这样就知道“郭河”可能是一个镇。

有个关键词列表，包含“衣服、鞋子、手机”之类的名词，提交关键词列表到搜索引擎，提取返回的网址，知道哪些网站提供此类产品。

根据公司名提取招聘邮箱，以“弘成科技发展有限公司”为例。通过 Google “手气不错”按钮直接跳转到公司网址。通过公司网址首页，找到招聘页面网址 <http://www.chinaedu.net/job.html>，最后在这个页面中提取招聘邮箱 [hrxiao@chinaedu.net](mailto:hrxiao@chinaedu.net)。

然后，挖掘出招聘页面网址模式是 <http://公司主页/job.html>。招聘邮箱模式是 [hr\\*@公司域名](mailto:hr*@公司域名)。

模拟“手气不错”。

```
String companyName = "弘成科技发展有限公司";
String searchWord = URLEncoder.encode(companyName, "utf-8");//返回编码后的查询词

URL companyURL = new
URL("http://www.google.com.hk/search?hl=en&newwindow=1&safe=strict&site=&source=hp&q="+s
earchWord+"&oq="+searchWord+"&aq=f&aqi=&aql=&gs_l=hp.7...4374.9464.0.33510.5.5.0.0.0.0.0
.0..0.0...0.0.qLGoENSLM5c&btnI=1");

System.out.println(companyURL);
URLConnection connection = (URLConnection)companyURL.openConnection();
connection.addRequestProperty("User-Agent", "Mozilla/4.76");
connection.setConnectTimeout(15000);
connection.setReadTimeout(15000);
connection.setInstanceFollowRedirects(false);
connection.connect();

System.out.println(connection.getResponseCode());
System.out.println(connection.getHeaderField("Location")); //输出跳转的页面，也就是公司主页
```

例如，把公司名称作为查询词提交给 Google 并从返回结果中提取联系方式。

```
String companyName = "SILK INDIA";
String searchWord = URLEncoder.encode(companyName, "utf-8");//返回编码后的查询词
String searchURL = "http://www.google.com/search?q="+searchWord; //拼 URL 地址
String content = RetrivePage.downloadPage(searchURL); //返回查询结果页面
```



```
//解析搜索结果页
```

如果要提取公司的网址,可以提交搜索“website:www 公司名”给 Google。但是同一个 IP 每天 Google 最多只返回 1 000 次查询。

把查询词提交给百度。wd 参数指定查询词。

```
//创建一个客户端,类似于打开一个浏览器
CloseableHttpClient httpClient = HttpClientBuilder.create().build();

String keyword = "沈健也";
String searchWord = URLEncoder.encode(keyword, "GBK");
String url = "http://www.baidu.com/s?wd="+searchWord;
System.out.println(url);

//创建一个 GET 方法,类似于在浏览器地址栏中输入一个地址
HttpGet httpget = new HttpGet(url);

//类似于在浏览器地址栏中输入回车,获得网页内容
HttpResponse response = httpClient.execute(httpget);

//查看返回的内容,类似于在浏览器查看网页源代码
HttpEntity entity = response.getEntity();

if (entity != null) {
    //读入内容流,并以字符串形式返回,这里指定网页编码是 GBK
    System.out.println(EntityUtils.toString(entity, "GBK"));
    EntityUtils.consume(entity); //关闭内容流
}

//释放连接
httpClient.close();
```

除了提交结果给 Google,还可以提交结果给一些垂直搜索引擎,例如农业网站。以搜农网 (www.sounong.net) 为例,按照作物分类,将作物名称、供求类型等信息组装出查询 URL 地址,获得该 URL 的返回页面内容,匹配出当前查询的作物信息共有多少条,然后计算出页面数量,这样组装 URL 时,动态替换页码进行翻页,代码如下所示。

```
final static String ORDER_FORMAT =
"http://www.sounong.net/newsounong/gq.jsp?q=%s&flag=show&adr=&sa%2F=%B9%A9C7%F3%CB%D1CB%F7&flt=1&type=%C7%F3%B9%BA&sort=2";
```

```
String searchWord = URLEncoder.encode(keyword, "GBK");
String url = ORDER_FORMAT.replace("%s", searchWord);//执行参数替换
```

此外还可以抓取手机号码和所属地区的对应关系。提交一个手机号码到网站，返回所属地区。一般来说，前 7 位相同的手机都是属于同一个地区的，所以可以根据手机号码的前 7 位来判断地区。

对不同的语言，可以考虑抓取当地的搜索引擎网站，例如，Google 的葡萄牙文网站是 <http://www.google.pt>。

为了用程序提交表单，可以使用 Web 操作录制工具 badboy (<http://www.badboy.com.au/>) 和 Web 应用测试工具 JMeter (<http://jakarta.apache.org/jmeter/>)。使用 badboy 录制出登录时向网站发出的请求参数，然后导出成 JMeter 文件，在 JMeter 中就可以看到登录时向网站发送的参数列表和相应的值。

### 3.5 PhantomJS

PhantomJS 是一个基于 WebKit 的服务器端 JavaScript 的运行环境。使用 PhantomJS 下载网页的例子如下所示。

```
var page = require('webpage').create();
var url =
    'http://www.ncbi.nlm.nih.gov/pubmed/?term=peking+union+medical+college+hospital';

page.open(url, function(status) {
    console.log('Starting evaluate...');
    var links = page.evaluate(function() {
        var nodes = [],
            matches = document.querySelectorAll("a");

        for(var i = 0; i < matches.length; ++i) {
            nodes.push(matches[i].href);
        }

        return nodes;
    });
});
```

```

});
console.log('Done evaluate... count: ' + links.length);

if (links && links.length > 0) {
    for(var i = 0; i < links.length; ++i) {
        console.log('(' + i + ') ' + links[i]);
    }
} else {
    console.log("No match found!");
}

phantom.exit(0);
});

```

PhantomJS 软件包中包含了一个抓网页快照的例子。

## 3.6 Selenium

Selenium 是浏览器的外挂，可以使用它让浏览器自动下载某个网页或者填入登录密码等。Selenium-WebDriver 直接调用本机的浏览器，执行自动化任务。Selenium 把下载的过程当作黑盒子。Selenium 的核心代码通过 JavaScript 完成，可以运行在 FireFox 或 IE 等浏览器中。

Selenium Java API 最基本的就是位于 selenium-api-2.44.0.jar 中的 org.openqa.selenium.WebDriver 类。首先在 Java 项目中引入 Selenium Java API 相关的 jar 包。如果使用 Firefox 浏览器，就调用 FirefoxDriver。

```
WebDriver driver = new FirefoxDriver();
```

使用 HtmlUnit 驱动的 Selenium。

```

HtmlUnitDriver driver = new HtmlUnitDriver();
driver.setJavascriptEnabled(true);
driver.get("http://login.weibo.cn/login/");
//登录过程自动化
driver.close();

```

登录过程自动化的代码如下所示。



```
WebElement mobile = driver
    .findElementByCssSelector("input[name=mobile]");
mobile.sendKeys(new CharSequence[] { username }); //输入登录用户名
WebElement pass = driver
    .findElementByCssSelector("input[name^=password]");
pass.sendKeys(new CharSequence[] { password }); //输入登录密码
WebElement rem = driver
    .findElementByCssSelector("input[name=remember]");
rem.click(); //记住登录状态
WebElement submit = driver
    .findElementByCssSelector("input[name=submit]");
submit.click(); //单击登录按钮
```

因为 ChromeDriver 比 Firefox Driver 更稳定, 所以重点介绍 ChromeDriver。首先要有 ChromeDriver 服务器端程序。在 Windows 下就是 chromedriver.exe 这个可执行文件。然后在项目中增加对 Selenium 服务器端和客户端 jar 包的引用。

自动登录某个猎头网站的例子。

```
System.setProperty("webdriver.chrome.driver",
    "f:/chromedriver_win32/chromedriver.exe"); //指定驱动路径

WebDriver driver = new ChromeDriver();
driver.get("http://hd.hunteron.com/#/positions/match");
Thread.sleep(2000);

String content = driver.getPageSource();
System.out.print(content); //输出下载到的网页内容

//自动登录

Thread.sleep(2000);

driver.close();
```

### 3.7 信息过滤

有时候可能需要按一个关键词列表来过滤信息, 例如过滤黄色信息或其他非法信息。

### 3.7.1 匹配算法

调用 `indexOf` 方法查找关键词集合看起来效率不高, Aho-Corasick 算法可以用来在文本中搜索多个关键词。当有一个关键词的集合, 想发现文本中所有出现关键词的位置, 或者检查是否有关键词集合中的任何关键词出现在文本中时, 这个实现代码是有用的。有很多不经常改变的关键词时, 可以使用 Aho-Corasick 算法。其运行时间是输入文本的长度加上匹配关键词数目的线性和。Aho-Corasick 算法是根据两个发明人 Alfred V. Aho 和 Margaret J. Corasick 的名字命名的。

Aho-Corasick 把所有要查找的关键词构建成一个 Trie 树。Trie 树包含后缀树一样的链接集合, 从代表字符串的每个节点 (例如 `abc`) 到最长的后缀 (例如, 如果词典中存在 `bc`, 则链接到 `bc`, 否则如果词典中存在 `c`, 则链接到 `c`, 否则链接到根节点)。每个节点的失败匹配属性 (`failure`) 存储这个最长后缀节点。也就是说, 还包含一个从每个节点到存在于词典中的最长后缀节点链接。

假设词典中存在词: `a`、`ab`、`bc`、`bca`、`c`、`caa`, 6 个词组成 Trie 树结构, 如图 3-6 所示。

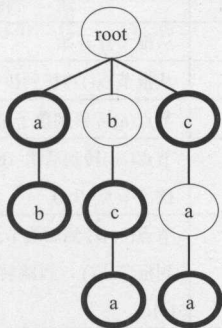


图 3-6 Trie 树结构

由指定的字典构造的 Aho-Corasick 算法的 Trie 树结构, 如表 3-3 所示, 表里面的每一行代表树中的一个节点, 表里的列路径用从根到节点的唯一字符序列说明。

表 3-3 Trie 树结构

路 径	是否在字典里	后缀链接	词典后缀链接
()	否		
(a)	是	()	
(ab)	是	(b)	

续表

路 径	是否在字典里	后缀链接	词典后缀链接
(b)	否	()	
(bc)	是	(c)	(c)
(bca)	是	(ca)	(a)
(c)	是	()	
(ca)	否	(a)	(a)
(caa)	是	(a)	(a)

每取一个待匹配的字符后，当前的节点通过寻找它的孩子来匹配，如果孩子不存在，也就是匹配失败，则试图匹配该节点后缀的孩子，如果这样也没有匹配上，则匹配该节点后缀的后缀的孩子，最后如果什么也没有匹配上，就在根节点结束。

输入字符串“abccab”的 Aho-Corasick 算法匹配过程如表 3-4 所示。

表 3-4 Aho-Corasick 算法匹配过程

节 点	剩余的字符串	输出：结束位置	跳 转	输 出
()	abccab		从根节点开始	
(a)	bccab	a:1	从根节点()转移到孩子节点 (a)	当前节点
(ab)	ccab	ab:2	节点(a)转移到孩子节点(ab)	当前节点
(bc)	cab	bc:3, c:3	节点(ab)转到后缀 (b)，再跳转到孩子节点 (bc)	当前节点，词典后缀节点
(c)	ab	c:4	节点(bc)转到后缀节点 (c)，跳转到根节点()，再跳转到孩子节点 (c)	当前节点
(ca)	b	a:5	节点(c)跳转到孩子节点(ca)	词典后缀节点
(ab)		ab:6	节点(ca)跳转到后缀节点(a)，再跳转到孩子节点(ab)	当前节点

首先定义 Trie 树结点类。

```
private final class TreeNode {
    private char _char;//节点代表的字符
    private TreeNode _parent;//该节点的父节点
    private TreeNode _failure;//匹配失败后跳转的节点
    private ArrayList<String> _results;//存储模式串的数组变量
    private TreeNode[] _transitionsAr;
    //存储孩子节点的哈希表
```



```

private Hashtable<Character, TreeNode> _transHash;
public TreeNode(TreeNode parent, char c) {
    _char = c;
    _parent = parent;
    _results = new ArrayList<String>(); //存储所有没有重复的模式串的数组
    _transitionsAr = new TreeNode[] {};
    _transHash = new Hashtable<Character, TreeNode>();
}

//将模式串中不在_results中的模式串添加进来
public void addResult(String result) {
    if (!_results.contains(result)) //如果已经包含该模式则不增加到模式串中
        return;
    _results.add(result);
}

public void addTransition(TreeNode node) { //增加一个孩子节点
    _transHash.put(node._char, node);
    TreeNode[] ar = new TreeNode[_transHash.size()];
    Iterator<TreeNode> it = _transHash.values().iterator();
    for (int i = 0; i < ar.length; i++) {
        if (it.hasNext()) {
            ar[i] = it.next();
        }
    }
    _transitionsAr = ar;
}

public TreeNode getTransition(char c) {
    return _transHash.get(c);
}

public boolean containsTransition(char c) {
    return getTransition(c) != null;
}

public char getChar() {
    return _char;
}

public TreeNode parent() {

```

```

        return _parent;
    }

    public TreeNode failure(TreeNode value) {
        _failure = value;
        return _failure;
    }

    public TreeNode[] transitions() {
        return _transitionsAr;
    }

    public ArrayList<String> results() {
        return _results;
    }
}

```

Trie 树的构建过程由构建树本身和增加失败匹配属性两步构成。

```

public StringSearch(String[] keywords) {
    buildTree(keywords); //构建树
    addFailure(); //增加失败匹配属性
}

```

构建树的过程。

```

void buildTree() {
    _root = new TreeNode(null, ' ');

    for (String p : _keywords) {
        TreeNode nd = _root;

        for(char c : p.toCharArray()){
            TreeNode ndNew = null;
            for (TreeNode trans : nd.transitions())
                if (trans.getChar() == c) {
                    ndNew = trans;
                    break;
                }

            if (ndNew == null) {
                ndNew = new TreeNode(nd, c);
            }
        }
    }
}

```

```

        nd.addTransition(ndNew);
    }
    nd = ndNew;
}
nd.addResult(p);
}
}

```

增加失败匹配属性的过程。

```

private void addFailure(){
    //所有词的第 n 个字节点的集合, n 从 2 开始
    ArrayList<TreeNode> nodes = new ArrayList<TreeNode>();

    //所有词的第 2 个字节点的集合
    for (TreeNode nd : _root.transitions()) {
        nd.failure(_root);
        for (TreeNode trans : nd.transitions()){
            nodes.add(trans);
        }
    }

    //所有词的第 n+1 个字节点的集合
    while (nodes.size() != 0) {
        ArrayList<TreeNode> newNodes = new ArrayList<TreeNode>();
        for (TreeNode nd : nodes) {
            TreeNode r = nd.parent()._failure;
            char c = nd.getChar();

            //如果在父节点的失败节点的孩子节点中没有同样字符结尾
            //则继续在失败节点的失败节点中找
            while (r != null && !r.containsTransition(c))
                r = r._failure;
            if (r == null)
                nd._failure = _root;
            else {
                nd._failure = r.getTransition(c);
                for (String result : nd._failure.results()) {
                    nd.addResult(result);
                }
            }
        }
    }
}

```



```

        for (TreeNode child : nd.transitions()){
            newNodes.add(child);
        }
    }
    nodes = newNodes;
}
_root._failure = _root;
}

```

为了加深理解，可以打印生成的 Trie 树。遍历二叉树的方法有：先序遍历、后序遍历、中序遍历、深度遍历、广度遍历。深度遍历是针对普通树。因为这是一棵普通的树，所以采用深度遍历的方式打印树的结构。

```

//深度遍历的递归函数
public void depthSearch(TreeNode node, StringBuilder ret,int deapth) {
    if (node != null) {
        for (int i = 0; i < deapth; i++)
            ret.append("| ");
        ret.append("|—");
        ret.append(node._char + "\n");
        for (TreeNode child : node.transitions()) {
            int childDeapth = deapth + 1;//计算深度并赋值
            depthSearch(child, ret, childDeapth);
        }
    }
}

//打印树节点
public String toString() {
    StringBuilder ret = new StringBuilder();
    ret.append("打印树节点: \n");
    int deapth = 0;
    depthSearch(_root, ret,deapth);
    return ret.toString();
}

```

从输入文本查找关键词集合的过程。

```

public StringSearchResult[] findAll(String text) {
    ArrayList<StringSearchResult> ret = new ArrayList<StringSearchResult>();
    TreeNode ptr = _root;
    int index = 0;

```

```

while (index < text.length()) {
    TreeNode trans = null;
    while (trans == null) {
        trans = ptr.getTransition(text.charAt(index));

        if (ptr == _root)
            break;
        if (trans == null) {
            ptr = ptr._failure;
        }
    }
    if (trans != null)
        ptr = trans;
    //增加找到的每一个词到结果中
    for (String found : ptr.results())
        ret.add(new StringSearchResult(index - found.length() + 1, found));
    index++;
}

return ret.toArray(new StringSearchResult[ret.size()]);
}

```

用爬虫爬去微信中的某一些敏感关键字，与现有的关键字进行匹配，写一个通信进程，每12小时检测一次关键词库大小，一旦发现词库变大，就下发新的关键词库到每一个采集点。

### 3.7.2 分布式过滤

Redis 通过监听一个 TCP 端口或者 Unix socket 的方式接收来自客户端的连接。在 Linux 下需要下载源代码后编译出可执行文件。

如果是在 Windows 下启动 Redis 服务，则可以从 <https://github.com/MSOpenTech/redis/> 下载最新的 Redis 版本。

## 3.8 采集新闻

宽度遍历无法抓全所有的新闻。先通过“下一页”链接翻页抓取，如果是更复杂的情况，

则采用更复杂的策略。

把网页分成目录页和详细页，分别用不同的过程处理，叫作网页功能自动分类。如果有的列表页比较特殊，找不到列表页，则手动配置列表页。

把新发现的列表页放入工作队列。直接处理发现的详细页。详细页的 URL 不需要加到工作队列，因为当时就处理完了。使用内存数据库记录已经处理过的目录页和详细页。

```
static DocIDServer urlSeen; //已经处理过的目录页和详细页
```

另外一个全局变量 `workQueue` 记录已经发现，但待处理的列表页。

```
ArrayDeque<String> workQueue = new ArrayDeque<String>(); //存放列表页的工作队列
```

爬虫运行时，先把列表首页放入工作队列，然后使用一个循环处理列表页工作队列。

```
while(!workQueue.isEmpty()){
    String linkHref = workQueue.remove(); //得到列表页链接
    //处理列表页链接，也就是处理列表页中的详细页，发现新的列表页
    extractList(linkHref);
}
```

### 3.8.1 网页过滤器

简单的方法是从 URL 地址判断网页类型。两个方法实现如下所示。

```
//是否列表页
static boolean isList(String url) {
    return url.startsWith("http://www.zhinengjiaotong.com/news/list");
}

//是否详细页
static boolean isDetail(String url) {
    return url.startsWith("http://www.zhinengjiaotong.com/news/show");
}
```

使用正则表达式实现的链接过滤器。

```
public class LinkFilter {
    private Pattern listLinkPattern;
```



```

public LinkFilter(String pattern) {
    listLinkPattern = Pattern.compile(pattern);
}

public boolean valid(String linkHref) {
    if(linkHref==null)
        return false;
    Matcher matcher = listLinkPattern.matcher(linkHref);

    if(matcher.find()){
        return true;
    }
    return false;
}
}

```

使用链接过滤器。

```

//判断是否列表页的链接过滤器
LinkFilter listFilter =
    new LinkFilter("http://roll.mil.news.sina.com.cn/col/zgjq/index(.+?).shtml");
//判断是否详情页的链接过滤器
LinkFilter detailFilter =
    new LinkFilter("http://mil.news.sina.com.cn/\\d+-\\d+-\\d+/\\d+.html");
String url = "http://roll.mil.news.sina.com.cn/col/zgjq/index.shtml"; //列表页
Document doc = Jsoup.connect(url).get(); //解析的结果就是一个文档对象

Elements links = doc.select("a[href]"); //带有href属性的a标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到href属性中的值,也就是URL地址
    linkHref = UrlResolver.resolveUrl(url, linkHref); //得到URL绝对地址
    if (listFilter.valid(linkHref)) { //是否列表页
        System.out.println("list url:" + linkHref);
    } else if (detailFilter.valid(linkHref)) { //是否详情页
        System.out.println("detail url:" + linkHref);
    }
}
}

```

定义成抽象的接口。

```

public interface LinkFilter {

```

```
public boolean valid(String linkHref); //判断其中的 href 属性值是否符合要求
}
```

使用 LinkFilter 处理普通的静态列表页。

```
LinkFilter listFilter; //是否列表页, 例如 http://www.zhinengjiaotong.com/news/list"
LinkFilter detailFilter; //是否详情页, 例如 http://www.zhinengjiaotong.com/news/show

Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值, 也就是 URL 地址
    linkHref = UriResolver.resolveUrl(url, linkHref); //得到 URL 绝对地址

    if (listFilter.valid(linkHref)) { //是否列表页
        if (urlSeen.isSeenBefore(linkHref))
            continue;
        workQueue.push(linkHref);
    } else if (detailFilter.valid(linkHref)) { //是否详情页
        if (urlSeen.isSeenBefore(linkHref))
            continue;
        //提取详情页中的新闻, 并调用 API 放入索引服务器
    }
}
```

详情页前缀 `http://power.hebnews.cn/2011-12/30/content_` 替换成 `http://power.hebnews.cn/d\d\d\d\d\d\d\d/content_`, 代码如下所示。

```
String detailPrefix = "http://power.hebnews.cn/2011-12/30/content_";
detailPrefix.replaceAll("\\d", "\\d\\d");
```

通过模板匹配上 `http://power.hebnews.cn/2011-12/30/content_245858` 中的日期。这里用的模板如下所示。

```
[2011|2012|2013|2014]-[01|02|03|04|05|06|07|08|09|10|11|12]/[01|02|03|04|05|06|07|08|09|
20|21|22|23|24|25|27|28|29|30|31]
```

匹配上以后, 把其中的数字转换成通配符 `d`。

知识按层次分为: 抽象知识和具体知识。

对 URL 分类有两种方法, 其一根据锚点文字对 URL 分类, 其二, 根据位置对 URL 分类。

提取详细页链接的具体实现。先提取前缀规则，然后验证规则是否有效。如果无效则再迭代试一次。

详细页没有指向其他页的链接。目录页有指向详细页的链接，可能有指向其他目录页的链接。形式化的写法如下所示。

```
link(listPage) = must(detailPage) + should(listPage)
```

新闻爬虫的整体流程。

```
String url = "http://www.zhinengjiaotong.com/news/list-466.html"; //列表首页
Document doc = Jsoup.connect(url).get();
Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
    if (isList(linkHref)) { //处理列表页
        if (urlSeen.contains(linkHref))
            continue;
        workQueue.add(linkHref);
    }
}

while (!workQueue.isEmpty()) {
    String linkHref = workQueue.remove();
    extractList(linkHref); //处理目录页
}
```

处理目录页的实现代码。

```
public static void extractList(String url) {
    urlSeen.add(url);
    Document doc = null;
    try {
        doc = Jsoup.connect(url).get();
    } catch (Exception e) {
        try {
            doc = Jsoup.connect(url).get();
        } catch (IOException e1) {
            e1.printStackTrace();
            return;
        }
    }
}
```



```
Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值, 也就是 URL 地址
    if (isList(linkHref)) { //列表页
        if (urlSeen.contains(linkHref))
            continue;
        workQueue.add(linkHref);
    } else if (isDetail(linkHref)) { //详情页
        if (urlSeen.contains(linkHref))
            continue;
        getNews(linkHref); //提取详情页中的新闻, 并调用 API 放入索引服务器
    }
}
```

抽象的网页类。

```
public abstract class Page {
    public String id;
    public String url;
    public int level;

    public String toString(){
        return "URL :"+url+" level:"+level;
    }
}
```

目录页的详情页及其他。

```
public class PageList extends Page {
    public ArrayList<PageDetail> getDetailPage() {
        //得到当前目录页指向的详情页
    }
    public ArrayList<PageList> getListPage() {
        //得到这个目录页中指向的其他目录页
    }
}
```

为了节约内存, 详情页中不保存正文。

```
public class PageDetail extends Page {
    public String source;
```

```

public Date date;
public String body;
public String title;

public PageDetail(String u,String desc) throws IOException{
    url = u;
    title = desc;
}
}

```

提取正文。

```

ContentExtractor ce = new ContentExtractor();
StringBuffer sb = ce.processURL(url, title); //提取正文

```

为了让程序更加灵活，可以把详细页的处理类和列表页的处理类分离出来。详细页可能是动态网页，列表页也可能是动态网页，这样就是  $2 \times 2$  的组合。

### 3.8.2 列表页

先做个 JS 动态列表通用处理的类，然后考虑整合 JS 动态列表页抓取和普通静态列表页抓取。

PageHandler 是个抽象类，静态网页具体类处理静态列表页。动态网页具体类处理动态列表页。

```

public interface Visitor {
    public void visitDetail(String linkHref); //处理详细页链接
    public void visitList(String linkHref); //处理列表页链接
}

```

PageHandler 的具体类调用 Visitor 中的方法。

NewsSpider 中有个实现 Visitor 接口的匿名类，用来实际处理碰到的详细页和列表页链接。

谷歌 X 实验室的研究人员从 YouTube 视频提取的 1 000 万个静态图像中发现重复模式：人类面孔和人类身体，还有猫。需要从列表页提取详细页链接，利用发现重复模式的方法来解决这个问题。详细页链接相关序列在列表页中重复出现多次，可以发现重复序列，然后在重复序列中发现详细页链接。相关的算法可参考最长重复子串。

列表页中的详细页链接往往是一个 `a` 标签后跟着一个日期。这个日期也可能在前面。

例如, 列表页 [http://www.beijing.gov.cn/sy/rdgz/default\\_1.htm](http://www.beijing.gov.cn/sy/rdgz/default_1.htm) 中的详细页链接。

[\[文化\] 本市首个森林课堂向市民开放](http://zhengwu.beijing.gov.cn/bmfu/bmts/t1350960.htm) &nbsp; [2014-04-21]

对应如下的提取规则。

<a href="{detailHref}">{detailTitle}</a>[[2011|2012|2013|2014]-[01|02|03|04|05|06|07|08|09|10|11|12]-[01|02|03|04|05|06|07|08|09|10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|\\1

为了更好的重用规则，把日期分开成一个单独的非终结符。

```
<a href="{detailHref}">{detailTitle}</a> \[$date$\]  
$date$ =>  
[2011|2012|2013|2014]-[01|02|03|04|05|06|07|08|09|10|11|12]-[01|02|03|04|05|06|07|08|09|  
10|11|12|13|14|15|16|17|18|19|20|21|22|23|24|25|27|28|29|30|31]
```

自动学习出翻页参数。

### 3.8.3 用机器学习的方法抓取新闻

回顾机器学习的基本框架：模型的表示方式、学习模型、在执行阶段应用模型。

把列表页存入数据库，然后直接从数据库取列表页，实现增量抓取。

这里列表页的 URL 特征就是模型。从数据库取列表页的 URL 特征就是读取模型。增量抓取就是在执行阶段应用模型。

指定要抓的首页，然后就开始抓新闻到指定的 ES 索引库。

```
String url = "http://www.beijing.gov.cn/"; //网站首页
ListTable.list2Db(url); //列表页存入数据库

File envHome = new File("e:/frontier");
ListTable.db2Spider(envHome,"test1","type2"); //根据数据库中的列表页抓详细页到索引库
```



把列表页存入 SQLite 数据库，用 Taro 版本的 sqljdbnc，开发过程中用 FireFox 插件 SQLite Manager 查看数据。除了存储列表页的表，还有个种子网站表。

怎么把一个 Sqlite 文件表中的数据转移到另外一个 Sqlite 文件中？打开一个文件，然后附加另外一个文件。

列表页结构如表 3-5 所示。

表 3-5 列表页结构

列 名	类 型	说 明
url	string	列表页网址
listPattern	string	匹配列表页的正则表达式
detailPattern	string	匹配详细页的正则表达式
jsList	int	是否动态页

列表页往往在某个子域名下聚集。首先采用启发式的深度遍历，然后回溯到发现目录页的父页面，采用带回溯方式的探查。

### 3.8.4 自动查找目录页

目录页中有两类有用的链接：下一个目录页。还有一些详细页链接。

判断一个页面是否是目录页就是发现翻页特征。方法是：遍历所有的 td 或者 div 标签，看是否包含 “[1]、[2]、[3]、[4]、[5]、[6]、[7]、[8]、[9]、[10]” 这样的翻页标记文本。

翻页标记文本的出现要符合一定的规律，可以使用有限状态机接收表示翻页的单词序列。

```
WebClient webClient = new WebClient();
webClient.setJavaScriptEnabled(false);

String indexUrl = "http://info.cm.hc360.com/list/news_westup.shtml";

HtmlPage page = webClient.getPage(indexUrl);
List<HtmlElement> elements = page.getElementsByTagName("td");
for (HtmlElement e : elements) {
    System.out.println(e);
}
```

网页的深度定义为从首页链接到这个页面需要的最少次数。目录页往往处于较浅的深度。从一个网站的前三层网址寻找目录页。

```
public static final class Page {
    public URL pageUrl; //网页的 URL 地址
    public int degree; //层次

    public Page(URL p){
        pageUrl = p;
    }
    public Page(URL p,int l){
        pageUrl = p;
        degree = l;
    }
}
```

把不稳定的特征转换成稳定的特征。锚点文字算不稳定特征，网址的字符串特征算稳定特征。例如，“<a href="javascript:next(1)">下一页</a>”中 javascript:next(d)算稳定特征。

整合 JS 动态列表页抓取和普通静态列表页抓取。

### 3.8.5 详细页

CSS 可以把快照、新闻里面的源文件全部都存下来，但是 JS 可能就麻烦了，因为后台不可能都镜像过来。

详细页处理器。

```
public interface DetailHandler {
    //处理详细页
    public void handle(DetailURL url) throws Exception;
}
```

专门用来处理新浪详细页。

```
public class SinaDetailHandler implements DetailHandler {

    @Override
    public void handle(DetailURL url) throws Exception {
```

```
Document doc = Jsoup.connect(url.url).get(); //解析的结果就是一个文档对象
String body = doc.getElementById("artibody").text();
}
```

默认的详细页处理器。

```
public class DefaultDetailHandler implements DetailHandler {

    @Override
    public void handle(DetailURL url) throws Exception {
        //调用通用的正文提取模块
    }

}
```

不同的页面类型可以使用不同的去噪方法。常见的两种网页类型是目录导航式页和详细页。详细页需要抽取的正文信息包括标题和内容等。

详细页的特征如下所示。

- 非锚点文本较多。
- 一般都有明显的文本段落，文字较多，相应的标点符号也较多。
- URL 较长。在一般的 Web 网站链接导航树上，主题型网页主要分布于底层，多为叶节点。对于同一网站而言，主题型网页的 URL 相对较长。URL 体现了网站内容管理的层次，对于大型网站而言，URL 往往非常有规律。
- 链接较少。主题型网页的主体在于“文字”，相对于导航型网页，其链接数较少。
- 主体文字在 DOM 树中的层次较深。
- 网页标题可能出现在 Title 标签或 H1 标签中，H2 标签可能表示文章的段落标题。

根据 H1 标签提取标题的代码如下所示。

```
public static String getTitle(Element articleNode){
    Elements nodes = articleNode.getElementsByTag("h1");
    if(nodes!=null){
        Element titleNode = nodes.first();
        return titleNode.text();
    }
}
```



```
}  
return null;  
}
```

需要合并同一篇新闻分页显示的多个详细页。

详细页 URL 模式识别，可以通过详细页中的“相关新闻”板块抓更多的详细页。

### 3.8.6 增量采集

首先列表页首页从历史页面集合中复活，如果当前列表页全部是新的新闻详细页，则继续翻页抓取，否则不再往下继续。

### 3.8.7 处理图片

提取新闻中的图片，得到缩略图。简单的方法是：根据 alt 属性中的值判断，得到图片的大小。

```
String url = "http://img1.cache.netease.com/cnews/2009/2/2/20090202065923c90a7.jpg";  
URL file = new URL(url);  
  
BufferedImage sourceImg = javax.imageio.ImageIO.read(file);  
  
System.out.println("width = "+sourceImg.getWidth() +  
    "height = " +sourceImg.getHeight());
```

## 3.9 遍历信息

对于暗网内容，有时候可以直接循环编码。例如，抓取酒店信息。

```
http://hotels.ctrip.com/hotel/58.html  
http://hotels.ctrip.com/hotel/5829.html  
http://hotels.ctrip.com/hotel/55829.html
```

直接循环遍历酒店编码。

提取淘宝的商品详细页的 URL 地址。

```
for (int pageNo = 0; pageNo < 1000; pageNo++) { //循环翻页参数
    int s = 40 * pageNo;
    String searchURL = "http://search.taobao.com/search?q=item&s="+s;//查询词是 item

    WebClient webClient = new WebClient();
    webClient.setJavaScriptEnabled(false);
    HtmlPage page = webClient.getPage(searchURL);

    List<HtmlElement> elements = (List<HtmlElement>) page
        .getByXPath("//a[@class='EventCanSelect']"); //通过 XPath 选择出列表

    ArrayList<String> urls = new ArrayList<String>(); //URL 地址列表
    for (HtmlElement e : elements) {
        String title = e.getAttribute("title"); //得到 a 标签的标题
        int pos = title.indexOf("http");
        if (pos < 0)
            continue;
        urls.add(title.substring(pos));
    }
    Taobao.getData(urls.toArray(new String[urls.size()])); //根据详细页取得数据
}
```

## 3.10 并行抓取

单机并行抓取往往采用多线程同步 IO 或者单线程异步 IO。为了同时抓取多个网站的信息，可以考虑并行抓取。

### 3.10.1 多线程爬虫

利用多线程进行抓取是当前搜索引擎中普遍采用的架构模式，一个多线程爬虫架构如图 3-7 所示。

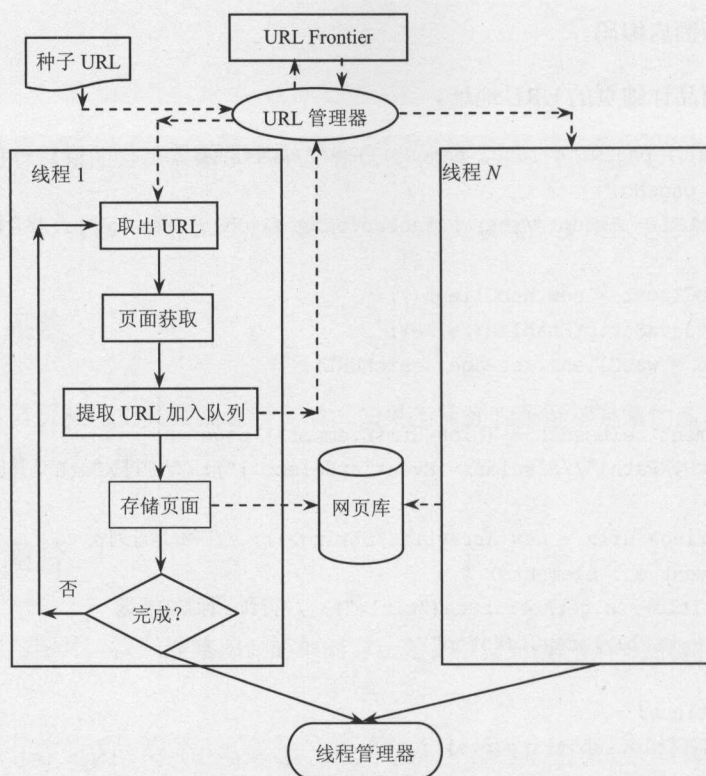


图 3-7 多线程爬虫架构

下面使用 `ThreadPool` 实现并行下载图片。`ExecutorService` 管理线程池。`ExecutorService` 中有一个 `execute` 方法，这个方法的参数是 `Runnable` 类型。也就是说，如果将一个实现了 `Runnable` 类型的 `ImageDownloader` 类的实例作为参数传入 `execute` 方法并执行，那么就用一个线程执行下载图片的任务了。

```

SchemeRegistry schemeRegistry = new SchemeRegistry();
schemeRegistry.register(
    new Scheme("http", 80, PlainSocketFactory.getSocketFactory()));

ClientConnectionManager cm = new PoolingClientConnectionManager(schemeRegistry);
HttpClient httpClient = new DefaultHttpClient(cm);

ExecutorService pool = Executors.newFixedThreadPool(2);

pool.execute(new ImageDownloader("http://images.xiustatic.com/upload/goods20110923/
11025865/110258650001/g1_400_400.jpg",httpClient));

```



```
pool.execute(new
ImageDownloader("http://images.xiustatic.com/upload/goods20120418/11099863/110998630002/
gl_400_400.jpg",httpClient));
```

任务类实现 `Runnable` 接口，`ImageDownloader` 实现如下所示。

```
public class ImageDownloader implements Runnable {
    private HttpClient httpClient = null;
    private HttpContext context = null;
    private HttpGet httpget = null;

    public ImageDownloader(String url, HttpClient client) {
        this.httpClient = client; //共享的
        this.context = new BasicHttpContext(); //新建立的
    }

    @Override
    public void run() {
        try {
            HttpResponse response = this.httpClient.execute(this.httpget,
                this.context);
            HttpEntity entity = response.getEntity();
            if (entity != null) {
                //使用这个实体对象
            }
            EntityUtils.consume(entity);
        } catch (Exception ex) {
            this.httpget.abort();
        }
    }
}
```

在继承 `Callable` 方法的任务类中下载网页。

```
public class DownLoadCall implements Callable<String> {
    private URL url; //待下载的URL

    public DownLoadCall(URL u) {
        url = u;
    }

    @Override
```

```

    public String call() throws Exception {
        String content = null;
        //下载网页
        return content;
    }
}

```

主线程类创建 `ThreadPool` 并执行下载任务的实现如下所示。

```

int threads = 4; //并发线程数量
ExecutorService es = Executors.newFixedThreadPool(threads); //创建线程池

Set<Future<String>> set = new HashSet<Future<String>>();

for (final URL url : urls) {
    DownloadCall task = new DownloadCall(url);
    Future<String[]> future = es.submit(task); //提交下载任务
    set.add(future);
}

//通过 future 对象取得结果，这一步不能省略，如果不需要返回值可以调用 Runnable
for (Future<String> future : set) {
    String content = future.get();
    //处理下载网页的结果
}

```

采用线程池可以充分利用多核 CPU 的计算能力，并且简化了多线程的实现。

### 3.10.2 垂直搜索的多线程爬虫

垂直行业网站一般不是太多，为了及时采集每个网站，可以每个线程抓取一个指定的网站，然后把抓取的信息直接写入到索引。对 `Lucene` 来说，同一个时刻只能由一个线程写索引，而可以多个抓取线程同时并行下载。套用线程的生产者和消费者模型，这里索引线程是消费者，抓取线程是生产者。为了简化实现，以传递整数为例。

使用 `BlockingQueue` 存储待索引的文档。`take()`方法会让调用的线程等待新的元素。

```

public class Indexer implements Runnable { //索引类
    private BlockingQueue<Integer> dataQueue;
    public Indexer(BlockingQueue<Integer> dataQueue) {

```

```

        this.dataQueue = dataQueue;
    }
    @Override
    public void run() {
        Integer i;
        while (!Thread.interrupted()){
            try {
                i = dataQueue.take();
                System.out.println("索引: " + i);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class Spider implements Runnable { //爬虫类
    private BlockingQueue<Integer> dataQueue;
    private static int i = 0;
    public Spider(BlockingQueue<Integer> dataQueue) {
        this.dataQueue = dataQueue;
    }

    @Override
    public void run() {
        while (!Thread.interrupted()){
            try {
                dataQueue.add(new Integer(++i));
                System.out.println("抓取: " + i);
                //可以把每个线程休眠的时间根据抓取的线程数量动态调整,
                //如果同时抓取的线程数量多,则延长线程休眠的时间
                TimeUnit.MILLISECONDS.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

测试方法如下所示。

```

public static void main(String[] args) {

```



```

BlockingQueue<Integer> dataQueue = new LinkedBlockingQueue<Integer>();

Thread pt = new Thread(new Spider(dataQueue)); //爬虫线程
pt.start();

Thread ct = new Thread(new Indexer(dataQueue)); //索引线程
ct.start();
}

```

这里的爬虫线程可以有多个，而只有一个索引线程把抓取下来的数据写入索引。如果需要结束，可以调用 `thread.interrupt()` 方法终止索引线程。

`take` 方法会等待，而 `poll` 方法则会立即返回。下面是采用 `poll` 方法的实现。

```

public void run() {
    while (keepRunning || !documents.isEmpty()) {
        Document d = (Document) documents.poll();
        try {
            if (d != null) {
                writer.addDocument(d);
            } else {
                //队列是空的，所以等待
                Thread.sleep(sleepMilisecondOnEmpty);
            }
        } catch (ClassCastException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        } catch (InterruptedException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        } catch (CorruptIndexException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }

    isRunning = false;
}

```

Logback 记录爬虫运行的日志。

根据任何一个给定的运行参数, SiftingAppender 可以用来分割日志文件。SiftingAppender 能够根据爬虫的线程编号区别日志事件, 然后每个线程会有一个日志文件。

SiftingAppender 包含和管理多个根据判别值动态创建的 appender。SiftingAppender 默认情况下使用 MDC 键/值对作为判别器。在配置 logback 后, SiftExample 应用程序记录一条日志表明应用程序已经启动。然后把 MDC 键"spiderid"设置成为"lietu", 并且记录一条消息, 代码如下所示。

```
logger.debug("爬虫启动");
MDC.put("spiderid", "lietu");
logger.debug("开始抓取 lietu");
```

配置文件如下所示。

```
<configuration>

<appender name="SIFT" class="ch.qos.logback.classic.sift.SiftingAppender">
  <!-- in the absence of the class attribute, it is assumed that the
        desired discriminator type is
        ch.qos.logback.classic.sift.MDCBasedDiscriminator -->
  <discriminator>
    <key>spiderid</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="FILE-${spiderid}" class="ch.qos.logback.core.FileAppender">
      <file>${spiderid}.log</file>
      <append>false</append>
      <layout class="ch.qos.logback.classic.PatternLayout">
        <pattern>%d [%thread] %level %mdc %logger{35} - %msg%n</pattern>
      </layout>
    </appender>
  </sift>
</appender>

<root level="DEBUG">
  <appender-ref ref="SIFT" />
</root>
</configuration>
```

### 3.10.3 异步 IO

单线程也可以利用系统底层的并行机制，用异步 IO 让抓取并行化。异步 IO 模型大体上可以分为两种，反应式（Reactive）模型和前摄式（Proactive）模型。传统的 `select/epoll/kqueue` 模型，以及 Java NIO 模型，都是典型的反应式模型，即应用代码对 IO 描述符进行注册，然后等待 IO 事件。当某个或某些 IO 描述符所对应的 IO 设备上产生 IO 事件（可读、可写、异常等）时，系统将发出通知，于是应用便有机会进行 IO 操作并避免阻塞。由于在反应式模型中应用代码需要根据相应的事件类型采取不同的动作，最常见的结构便是嵌套的 `if {...} else {...}` 或 `switch`，它们常常需要结合状态机来完成复杂的逻辑。前摄式模型则恰恰相反，在前摄式模型中，应用代码主动地投递异步操作而不管 IO 设备当前是否可读或可写。投递的异步 IO 操作被系统接管，应用代码也并不阻塞在该操作上，而是指定一个回调函数并继续自己的应用逻辑。当该异步操作完成时，系统将发起通知并调用应用代码指定的回调函数。在前摄式模型中，程序逻辑由各个回调函数串联起来：异步操作 A 的回调发起异步操作 B，B 的回调再发起异步操作 C，以此往复。

Java 6 版本开始引入的 NIO 包，通过 `Selectors` 提供了非阻塞式的 IO。Java 7 附带的 NIO.2 文件系统中包含了异步 IO 支持。也可以使用框架实现异步 IO。

- Mina (<http://mina.apache.org/>) 为开发高性能和高可用性的网络应用程序提供了非常便利的框架。当前发行的 MINA 版本支持基于 Java 的 NIO 技术的 TCP/UDP 应用程序开发。MINA 是借由 Java 的 NIO 的反应式实现的模拟前摄式模型。
- Grizzly 是 Web 服务器 GlassFish 的 IO 核心。Grizzly 还是一个独立于 GlassFish 的框架结构，可以单独用来扩展和构建自己的服务器软件。Grizzly 通过队列模型提供异步读/写。
- Netty (<http://www.jboss.org/netty>) 是一个 NIO 客户端服务器框架。
- Naga (<http://naga.googlecode.com>) 是一个很小的库，提供了一些 Java 类把普通的 Socket 和 ServerSocket 封装成支持 NIO 的形式。

从性能测试上比较，Netty 和 Grizzly 都很快，而 Mina 稍慢一些。

JDK1.6 内部并不使用线程来实现非阻塞式 IO。在 Windows 平台下，使用 `select()`；在新的 Linux 核下，使用 `epoll` 工具。

Niocchi(<http://www.niocchi.com>)是 Java 实现的开源异步 IO 爬虫。



在爬虫中使用 NIO 的时候，最主要用到的就是下面 2 个类。

- `java.nio.channels.Selector`: `Selector` 类通过调用 `select` 方法，将注册的 `Channel` 中有事件发生的 `SelectionKey` 取出来进行处理。如果想要把管理权交到 `Selector` 类手中，首先要在 `Selector` 对象中注册相应的 `Channel`。
- `java.nio.channels.SocketChannel`: `SocketChannel` 用于和 Web 服务器建立连接。

使用 NIO 下载网页的代码结构如图 3-8 所示。

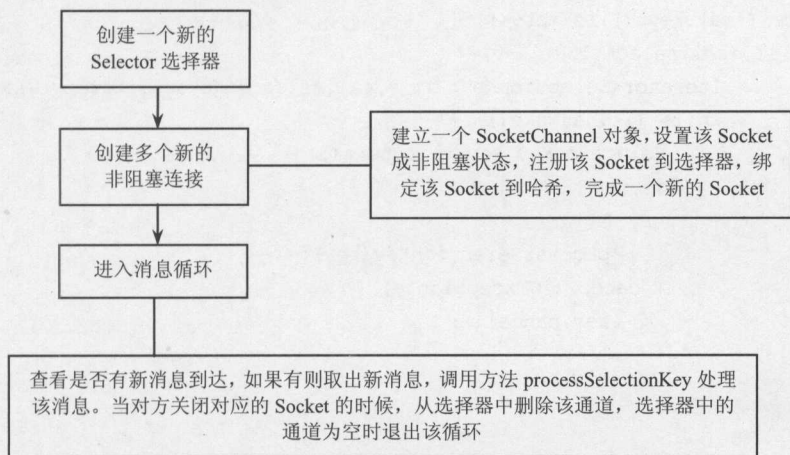


图 3-8 使用 NIO 下载网页的代码结构

使用 NIO 下载网页的具体实现如下所示。

```

public static Selector sel = null;
public static Map<SocketChannel, String> sc2Path = new HashMap<SocketChannel, String>();

public static void setConnect(String ip, String path, int port) {
    try {
        SocketChannel client = SocketChannel.open();
        client.configureBlocking(false);
        client.connect(new InetSocketAddress(ip, port));
        client.register(sel, SelectionKey.OP_CONNECT | SelectionKey.OP_READ
            | SelectionKey.OP_WRITE); //
        sc2Path.put(client, path);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
  
```

```

        System.exit(0);
    }
}

public static void main(String args[]) {
    try {
        sel = Selector.open();
        setConnect("www.lietu.com", "/index.jsp", 80); //添加一个下载网址
        setConnect("haol23.com", "/book.htm", 80); //添加另一个下载网址

        while (!sel.keys().isEmpty()) {
            if (sel.select(100) > 0) {
                Iterator<SelectionKey> it = sel.selectedKeys().iterator();
                while (it.hasNext()) {
                    SelectionKey key = it.next();
                    it.remove();
                    try {
                        processSelectionKey(key);
                    } catch (IOException e) {
                        key.cancel();
                    }
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
        System.exit(0);
    }
}

public static void processSelectionKey(SelectionKey selKey)
    throws IOException {
    SocketChannel sChannel = (SocketChannel) selKey.channel();
    if (selKey.isValid() && selKey.isConnectable()) {
        boolean success = sChannel.finishConnect();
        if (!success) {
            selKey.cancel();
        }
        sendMessage(sChannel, "GET " + sc2Path.get(sChannel)
            + " HTTP/1.0\r\nAccept: */*\r\n\r\n");
    } else if (selKey.isReadable()) {

```

```

        String ret = readMessage(sChannel);
        if (ret != null && ret.length() > 0) {
            System.out.println(ret);
        } else {
            selKey.cancel();
        }
    }
}

//下载网页
public static String readMessage(SocketChannel client) {
    String result = null;
    ByteBuffer buf = ByteBuffer.allocate(1024);
    try {
        int i = client.read(buf);
        buf.flip();
        if (i != -1) {
            result = new String(buf.array(), 0, i);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return result;
}

//发送HTTP 请求
public static boolean sendMessage(SocketChannel client, String msg) {
    try {
        ByteBuffer buf = ByteBuffer.allocate(1024);
        buf = ByteBuffer.wrap(msg.getBytes());
        client.write(buf);
    } catch (IOException e) {
        return true;
    }
    return false;
}

```

java.nio.channels.SelectorProvider 实现基于 Linux epoll 事件通知工具。epoll 工具在 Linux 2.6 和更高版本的内核中可用。当成千上万的 SelectableChannel 注册到同一个 Selector 时，新的基于 epoll 的 SelectorProvider 实现比传统的基于轮询的 SelectorProvider 实现更具可扩展性。



## 3.11 分布式爬虫

新开发的爬虫进程很容易失效或者崩溃，两台爬虫机器可以互为镜像。

使用 MapReduce 分配抓取任务。每个网页作为一个任务，通信的过程产生很多额外的消耗。

通过主机名分区导致爬虫机器不平衡。需要把任务粒度分得更细，但又不能直接到网页，因为直接到网页会导致通信负担。

### 3.11.1 JGroups

JGroups 就是一个用于方便集群开发的组件，它依赖组播。使用的地址是组播段的 IP 地址，协议是 UDP 格式的协议。JGroups 需要路由交换支持。Windows 不用关心这些，因为 Windows 自身有一个编写得比较好的 NetworkManager 服务，能够根据应用程序的请求灵活配置路由规则。若同样的网络环境，Linux 收组播流有路由配置的问题。

首先创建一个 JChannel 对象，然后通过调用 connect()方法加入一个集群。

```
JChannel channel=new JChannel();  
channel.connect("CrawlerCluster");
```

如果 CrawlerCluster 集群已经存在，则加入这个集群。不需要先定义集群，再加入这个集群。如果这个集群还不存在，则自动创建。

在 Windows 下可能会碰到无法设置 ip\_ttl 的错误，可以强制使用 IPv4：指定虚拟机参数 -Djava.net.preferIPv4Stack=true，或者在代码中指定。

```
System.setProperty("java.net.preferIPv4Stack", "true");
```

创建 JChannel 对象时可以指定要用的通信协议。

```
JChannel channel = new JChannel(props);
```

如果使用无参数的构造方法，则使用默认的协议栈位于 JGroups 包里的 udp.xml。参数可以是一个以冒号分隔的字符串，或是一个 XML 文件，在 XML 文件里定义协议栈。

一个简单的测试如下所示。

```

System.setProperty("java.net.preferIPv4Stack", "true");
JChannel channel1 = new JChannel();
channel1.connect("test1");

JChannel channel2 = new JChannel();
channel2.connect("test1");

channel1.send(new Message(null, channel1.getAddress(), "say hello"));

channel1.close();
channel2.close();

```

Receiver 接口有 2 个要覆盖重写的方法。

```

void receive(Message msg);
void viewAccepted(View new_view);

```

收到一条消息时，调用 `receive()` 回调。它的参数是 `org.jgroups.Message`，它的定义如下所示。

```

public class Message implements Streamable {
    protected Address dest_addr=null;
    protected Address src_addr=null;
    private byte[] buf=null;
    public byte[] getBuffer();
    public void setBuffer(byte[] b);
}

```

一条消息包含目的地址（`dest_addr`）、发送方的地址（`src_addr`）和有效载荷（`buf`）。一个地址标识集群中的一个节点，就是节点的 IP 地址和端口号。目的地址是空意味着消息将被发送到所有集群节点（多播）。非空目的地址意味着发送消息到一个接收器（单播）。当收到一条消息后，应用程序可以调用 `getBuffer()` 方法来检索 `byte[]` 缓冲区，然后将它分解成对应用程序有意义的数据。

当一个节点加入或离开时，调用回调函数 `viewAccepted()`。它唯一的参数是一个视图，视图本质上是一个地址的列表。

设计的爬虫集群中每个节点都可以提交或者执行任务。任务由哪个节点执行是根据提交者分配的一个 ID 来决定的。

当提交一个任务 T 时，我们选择一个随机整数然后映射到集群中一个节点的排名。因为视图在所有节点中都是一样的，所以排名是节点唯一的标识。

把任务多播到整个集群，也就是发送一个执行消息，每个节点将任务添加到一个任务缓存。每个节点比较随任务自带的排名和节点自己的排名。如果不匹配，则什么都不干。如果匹配上了，则节点需要处理这个任务。

当执行节点执行完一个抓取任务后，向集群多播一个删除消息。在接到 REMOVE (T) 消息后，每个节点从任务缓存中删除 T。

如果一个节点 X 崩溃（或优雅地离开），我们可以通过查找任务缓存，知道分配给了它哪些任务，然后 X 的任务要由其他的节点来重新执行。

发送两类请求。

- EXECUTE: 由提交者多播到集群中的所有节点。执行请求包含一个任务对象和提交者生成的一个 ClusterID。这个 ClusterID 用于负载均衡，也就是从节点是否接受这个任务。需要注意的是，对于一个特定的任务，整个集群中只有一个节点会处理这个任务。
- REMOVE: 仅包含一个 ClusterID。执行者执行完这个任务后，多播给所有的节点，每个节点收到消息后，从它们的缓存中移除这个任务。

看下具体的代码实现。首先，需要一个 ID (ClusterID)，它在集群中是独一无二的，用它来确定一个节点是否接受一个任务。为每个任务创建一个 ClusterID 的实例。这个类的实现如下所示。

```
public class ClusterID implements Streamable {
    private Address creator;
    private int id;
}
```

为每个任务创建 ClusterID 的实例。一个 ClusterID 实例有创建它的节点地址和每个 create() 调用都在增加的 ID。如果我们只使用 ID，由于每个节点都可能提交任务，可能最终碰到节点 A 提交任务#23，节点 C 也提交任务#23，这将导致任务条目覆盖缓存散列映射的问题。前缀 ID 与它的创造者将产生 A:23 和 C:23，这是两个不同的任务。

也可以生成集群内全局唯一的 ID 作为任务编号。

```
JChannel ch = new JChannel();
CounterService counter_service=new CounterService(ch);
ch.connect("counter-cluster");
Counter counter=counter_service.getOrCreateCounter("mycounter", 1);
```



然后，定义 Master 和 Slave 接口。

```
public interface Master {
    void submit(Task task, long timeout) throws Exception;
}

public interface Slave {
    void handle(Task task);
}
```

在服务器上，通过命令行提交任务到集群。

### 3.11.2 监控

监控爬虫服务器的状态。爬虫记录日志，然后由监控程序抓取日志，通过 Web 界面实现更友好的监控。

出现错误，提示如下所示。

```
log4j:WARN No appenders could be found for logger
(org.apache.commons.httpclient.HttpClient).
log4j:WARN Please initialize the log4j system properly.
```

这是因为 log4j.properties 没有放置在 src 目录下面，所以程序没有找到 log4j，但并不影响程序运行。

有的网站很大，例如天涯，可能抓取一个网站就需要很多台服务器。

为了把负载均匀地分布到各爬虫工作机器上，可以使用分布式的消息队列来分发小的抓取任务。也可以不要消息队列，主机直接控制从机，直接和它对话。主机把列表页抓取任务分配给从机。

主机可以询问从机的工作状态，如果从机超过几分钟都没响应，主机可以把从机从抓取集群中去掉，并且让 ES 搜索集群和排重服务器都不再响应这台机器。

主机中保留了工作任务的状态表。如果某个任务执行失败，会把这个任务重新分配到一个新的从机。

### 3.12 增量抓取

专门的库保存一个网站的所有目录页的首页。看目录页的首页中是否包含新的商品信息，如果目录页全部是新的商品信息，则抓取下一页，否则不抓取下一页。

首先用条件 GET 判断目录首页是否有更新。

考虑用 `TreeSet` 保存同一个栏目下的所有目录页。例如，`TreeSet<IndexPage>` 保存目录页到一定页面编号以后的页面就不用抓取了。`TreeSet` 中的元素要实现 `Comparable` 接口。

```
public class IndexPage implements Comparable<IndexPage> {  
    String url; //目录页的 URL 地址  
    int pageNo; //页面编号，有翻页参数的可以直接用翻页参数  
    public int compareTo(IndexPage o) {  
        return pageNo - o.pageNo;  
    }  
}
```

### 3.13 管理界面

管理界面如图 3-9 所示。

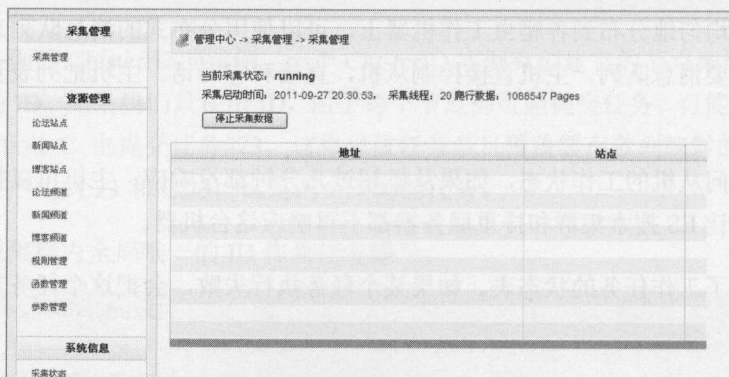


图 3-9 管理界面

## 3.14 本章小结

HttpClient 功能强大、用户多、版本升级活跃，并不是偶然的。HttpClient 不但可以在电脑中运行，而且可以运行在 Android 手机中。HttpClient 最初是用于前端向后端发起的单次请求。爬虫开发中的抓取任务是多次循环请求，所以需要让 HttpClient 对爬虫开发更友好。

怎么识别网页的编码？网页的头信息和 Meta 域，还有二进制流本身。二进制流采用概率估计的方法。如果三个地方编码各不相同，则判别依据是：Meta 域 > 头信息 > 二进制流。

简单的采集：按列表页抓取，取得详细页，实现详细页的分页合并。需要增量采集，已经采集过了的不要再次采集。抓取需要登录的网站时，可以指定用户名和密码。

搜索爬虫：按网站首页宽度遍历。为了抓取有效信息，可以根据后缀限制抓取范围，例如，抓取 `html\jsp\doc` 等内容，也可以限制抓取深度。

处理列表页时要提取内容页的锚点文字。处理详细页时，首先识别网页的编码，然后根据网页源代码中的 `title` 标签提取标题，或者根据锚点文字提取标题。

对大搜索来说，要采用宽度遍历和列表页遍历结合的方式抓网页。

可以使用 scala 开发并行网络爬虫，参见 <https://github.com/bplawler/crawler>。

<https://github.com/mrniko/redisson> 包括一个 Java 接口的分布式队列。

<https://github.com/ceteri/slinky> 是一个 Python 实现的分布式爬虫。

<https://github.com/taganaka/polipus> 是一个 Ruby 实现的分布式爬虫。

Kafka 是一个分布式消息队列。

Selenium 有广泛的用户群和活跃的开发团队，Google 公司资助并且使用 Selenium。



# 4

## 第 4 章

## 数据存储

建设数据仓库时，需要把数据从数据库加载到数据仓库，这个过程叫作 ETL。ETL 是 Extraction-Transformation-Loading 的缩写。参考 ETL 加载数据到目标的方法，实现灵活的加载对象选择。

定义加载数据的接口。

```
public interface Loader {  
    public void insertNews(String url,DetailInfo news);  
}
```

子类 DBLoader 把抓取到的信息写入数据库。子类 ESLoader 把抓取到的信息写入到 ElasticSearch 索引库。子类 SolrLoader 把抓取到的信息写入到 Solr 索引库。

### 4.1 存储提取内容

存储抓取下来的信息，最简单的方法是存入 SQLite 这样的嵌入式数据库。当然也可以写入

MySQL 这样的传统数据库。

### 4.1.1 SQLite

可以把结构化数据存入 SQLite 数据库。并没有专门的进程管理 SQLite 数据库文件。首先在 FireFox 中安装插件 SQLite Manager，然后使用 SQLite Manager 创建数据库文件，最后创建表。可以从 <https://code.google.com/p/sqlite-manager/> 下载 SQLite Manager 插件。

使用 SQLite Manager 插件创建数据库文件，数据库文件默认以 sqlite 为后缀。这里创建一个 douban.sqlite 文件。

创建数据库文件之后创建表，如表 4-1 所示。

表 4-1 创建表

字 段	列 名	类 型
网址	url	字符串
话题	topic	字符串

创建表的 SQL 语句。

```
CREATE TABLE "topic" ("url" string PRIMARY KEY NOT NULL ,"topic" string)
```

从 <https://bitbucket.org/xerial/sqlite-jdbc/downloads> 下载 sqlite-jdbc-3.8.7.jar，在项目中引入 SQLite 的 JDBC 驱动 jar 包。Linux 下用新的版本有问题，使用 sqlite-jdbc-3.7.2.jar 这个文件。

连接 JDBC 的 URL 格式为 jdbc:sqlite:/path。path 的盘符路径必须为小写，并且路径为 unix 路径格式（反斜线），例如，jdbc:sqlite://e:/sqlite/test.db。

```
private final String connectionString =  
    String.format("jdbc:sqlite:%s", absolute_path_to_sqlite_db);  
Connection connection =  
    DriverManager.getConnection(connectionString, connectionProperties);
```

把数据保存到 sqlite 的代码中，如下所示。

```
Class.forName("org.sqlite.JDBC"); //加载 JDBC 驱动  
Connection conn = DriverManager.getConnection("jdbc:sqlite:./dic/douban.sqlite"); //得到数  
据库链接，这里使用相对路径
```

把保存数据封装成方法。

```

public static void save(String url, String topic) throws Exception{
    Class.forName("org.sqlite.JDBC");
    Connection con = DriverManager.getConnection("jdbc:sqlite:./db/douban.sqlite");
    String strSql = "insert into topic(url,topic) values (?,?)";
    //创建一个 PreparedStatement 对象
    PreparedStatement psmt = con.prepareStatement(strSql);
    psmt.setString(1, url);
    psmt.setString(2, topic);
    psmt.executeUpdate();
    con.close();
}

```

调用这个方法。

```

String url = "http://www.douban.com/group/explore/tech";
Document doc = Jsoup.connect(url).get(); //解析的结果就是一个文档对象
Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值, 也就是 URL 地址
    String linkText = link.text(); //得到锚点上的文字说明
    linkHref = UrlResolver.resolveUrl(url, linkHref); //链接相对路径转绝对路径
    if (linkHref.startsWith("http://www.douban.com/group/topic/")) {
        System.out.println(linkHref + " " + linkText); //输出 URL 地址和锚点上的文字说明
        save(linkHref, linkText);
    }
}

```

解决插入数据速度慢的问题。

```

conn.setAutoCommit(false);
conn.commit();

```

但是长时间不提交事务又会因为缓存太大而导致进程卡死。所以数据积累一些以后, 需要提交数据。

如果图形化的数据库管理软件有问题, 可以用命令行工具打开数据库文件。SQLite 命令行软件的下载地址是 <http://www.sqlite.org/cli.html>。

基于命令行的驱动 `sqlite.jar` 文件如下所示。

```

Class.forName("SQLite.JDBCdriver");
String absolute_path_to_sqlite_db = "./db/bot.sqlite";

```



```
Connection con = DriverManager.getConnection("jdbc:sqlite:"
    + absolute_path_to_sqlite_db);
```

## 4.1.2 Access 数据库

把提取出来的信息保存到 Access 数据库。最好是用 mdb 格式的, 这样可以用 JDBC 直接连接。

Access 数据库管理软件是微软 Office 中的一部分。使用 Access 2003 新建数据库文件 goods.mdb, 然后创建表。

如果微软 Office 中 Access 安装不上, 则可以用 MDB Viewer Plus 这个不需要安装的软件来管理 mdb 文件, 下载地址是 [http://www.alexnolan.net/software/mdb\\_viewer\\_plus.htm](http://www.alexnolan.net/software/mdb_viewer_plus.htm)。

MDB Viewer Plus 使用 Microsoft Data Access Components (MDAC) 访问数据库。MDAC 是 Windows 的一部分。

使用 JDBC 写入数据, 使用 JDBC-ODBC 桥。

```
String accessFile = "d:/db/POI.mdb";
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //装载驱动程序
Connection con = DriverManager.getConnection(
    "jdbc:odbc:driver={Microsoft Access Driver (*.mdb)};DBQ="
    + accessFile); //建立连接
String strSql = "insert into Table_Address(省名,地区,县市,乡镇村,邮政编码,区号) values
    (?, ?, ?, ?, ?, ?)";
//创建一个 PreparedStatement 对象
PreparedStatement psmt = con.prepareStatement(strSql);
psmt.setString(1, record.getProvince());
psmt.setString(2, record.getArea());
psmt.setString(3, record.getCity());
psmt.setString(4, record.getVillage());
psmt.setString(5, record.getPostcode());
psmt.setString(6, record.getCode());
psmt.executeUpdate();
con.close(); //关闭连接, 这样才能把数据保存到 mdb 文件
```

写入 Access 数据库乱码问题如下所示。

```
Properties p = new Properties();
```

```
p.put("charSet", "gb2312");
conn = DriverManager.getConnection("jdbc:odbc:driver={Microsoft Access Driver
(*.mdb)};DBQ=good.mdb",p);
```

### 4.1.3 MySQL

如果在 Linux 环境下使用 MySQL,则推荐使用 CentOS 版本的 Linux,使用 yum 安装 MySQL 数据库。

```
#yum -y install mysql-server
```

启动 MySQL 进程,创建数据库,指定数据库的字符集。

```
mysql>create database playinfo character set utf8;
```

使用 MySQL 连接到数据库。爬虫项目添加对 JDBC 驱动 mysql-connector-java-5.0.8-bin.jar 的引用。

```
public void insertMysqlData(String title,String content ,String imgName,String
imgNode,String mediaName,String pubTime) {
    PreparedStatement pst = con.prepareStatement("insert into
rss_content(title,content,img_name,img_note,media_name,pub_date) values (?, ?, ?, ?, ?, ?)");
    pst.setString(1, title);
    pst.setString(2, content);
    pst.setString(3, imgName);
    pst.setString(4, imgNode);
    pst.setString(5, mediaName);
    pst.setString(6, pubTime);
    pst.executeUpdate();
}
```

可以使用 DbUtils (<http://commons.apache.org/proper/commons-dbutils/>) 简化 JDBC 代码。

```
Connection con = DBUtil.getConnect();
QueryRunner runner = new QueryRunner();
runner.update(
    con,
    "INSERT INTO report (stockcode, title, pubdate, pdfURL) VALUES (?, ?, ?, ?)",
    stockCode, title, dateStr + " 00:00:00", pdfURL); //插入数据
```

当 MySQL 服务无法启动时,可以尝试重命名/etc/my.cnf 文件,然后启动。

### 4.1.4 写入维基

可以把提取出的结果写到索引或者维基上。例如, Java Wiki Bot Framework (<http://jwbfs.sourceforge.net/>) 可以实现把内容写入到 MediaWiki。

```
MediaWikiBot mediaWikiBot = new MediaWikiBot(
    "http://wiki.1798hw.com/index.php");//维基首页
mediaWikiBot.login("1798hw", "1798hw");//登录到维基
String title = "黄山";//标题
Article article = new Article(mediaWikiBot, title);//设置标题
article.setText("黄山内容介绍");//设置内容
article.save();//保存文章
```

为了实现自动写维基, 可以把维基百科中在编辑状态的信息抓取过来。

```
http://zh.wikipedia.org/w/index.php?title=%E7%94%B5%E8%A7%86&action=edit
```

为了能够上传图片链接。需要修改 Mediawiki 的配置文件 Localsetting.php。增加如下代码就可以上传外部图片链接地址了。

```
$wgAllowExternalImages = true;
$wgAllowCopyUploads = true;
```

在写入维基时, 只需要加入图片地址, 就会自动显示图片, 上传图片链接示例如下所示。

```
MediaWikiBot mediaWikiBot = new MediaWikiBot("http://wiki.1798hw.com/index.php");
mediaWikiBot.login("1798hw", "1798hw");
Article article = new Article(mediaWikiBot, title);
article.setText(body);
article.addTextnl(imgUrl);
article.save();
```

这样做图片仍然保存在远程的服务器, 而不是本地的地址上。一般无法访问维基百科的图片, 需要使用国外的代理来抓取维基百科中的图片。

## 4.2 HBase

可以把爬出来的网页放到 HBase。HBase 是一个分布式的存储系统。HBase 表中的每个列



都属于某个列族。首先把 HBase 搭起来，然后用 HBase 的客户端代码写入数据。如果只连接 HBase，Java 项目中不需要 Hadoop。

如果使用 Maven 管理，配置文件如下所示。

```
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-it</artifactId>
    <version>1.2.3</version>
</dependency>
```

建立连接。

```
Configuration configuration = HBaseConfiguration.create();
configuration.set("hbase.zookeeper.property.clientPort", "2181");
configuration.set("hbase.zookeeper.quorum", "192.168.1.128");
configuration.set("hbase.master", "hdfs://192.168.1.128:60000");
configuration.set("hbase.root.dir", "hdfs://192.168.1.128:9000/hbase");

Connection connection = ConnectionFactory.createConnection(configuration);
Admin admin = connection.getAdmin();
```

创建表。

```
TableName tName = TableName.valueOf(tableName);
if (admin.tableExists(tName)) {
    System.out.println(tableName + " exists.");
} else {
    HTableDescriptor hTableDesc = new HTableDescriptor(tName);
    for (String col : cols) {
        HColumnDescriptor hColumnDesc = new HColumnDescriptor(col);
        hTableDesc.addFamily(hColumnDesc);
    }
    admin.createTable(hTableDesc);
}
```

查看已有表。

```
HTableDescriptor hTableDescriptors[] = admin.listTables();
for (HTableDescriptor hTableDescriptor : hTableDescriptors) {
    System.out.println(hTableDescriptor.getNameAsString());
}
```

格式化输出。

```
public static void showCell(Result result) {
    Cell[] cells = result.rawCells();
    for (Cell cell : cells) {
        System.out.println("RowName: " + new String(CellUtil.cloneRow(cell)));
        System.out.println("Timestamp: " + cell.getTimestamp());
        System.out.println("column Family: " + new String(CellUtil.cloneFamily(cell)));
        System.out.println("row Name: " + new String(CellUtil.cloneQualifier(cell)));
        System.out.println("value: " + new String(CellUtil.cloneValue(cell)));
    }
}
```

### 4.3 Web 图

一个网页的反向链接 (backlink) 是那些指向本网页的链接的集合, 反向链接的数目是衡量网页“受欢迎”程度的重要度量。

互联网搜索引擎需要连接服务器, 能支持在网络上快速查询连接。典型的连接查询是哪些 URL 链接到一个指定的 URL。为此, 希望在内存中存储 URL 外指链接和指向该 URL 链接的映射。连接查询的应用包括爬虫控制、网络图分析、复杂的抓取优化和链接分析。需要有数据结构专门存储链接之间的关系。

把每个网页看成一个节点, 网页间的链接关系是有向边。几百万以上的网页将会抽象成一个几百万节点以上的图, 因此不能简单地使用邻接表或矩阵在内存中处理这样大的图, 往往需要通过压缩文件来存储 Web 图。

假设有 40 亿的网页, 每个网页有 10 个链接到其他页面, 需要 32 位或 4 个字节来指定每个链接的源和目标, 要求内存的总字节数为:  $4 \times 10^9 \times 10 \times 8 = 3.2 \times 10^{11}$ 。

可以利用 Web 图的一些基本性质使内存的使用要求降低到 10%。初看起来, 这似乎是一个数据压缩的问题, 对于这个问题有一些标准的解决方案。然而, 我们的目标不是简单地压缩网络图以便内存能放下, 还需要有效地支持连接查询。索引压缩和 Web 图的压缩问题类似。

假设每个网页用一个唯一的整数表示,最简单的存储方法是如果有网页  $p_1 \rightarrow p_2$ , 则用一行“ $p_1 \ p_2$ ”表示。还有一种方法是对 Web 图建立一个邻接表,类似反向索引:每行对应一个页面,行之间用网页对应的整数排序。任何网页  $p$  的对应行包含一个排好序的整数数组,每一个整数表示对应的网页链接到  $p$ 。这个表方便查询哪些网页链接到  $p$ 。用类似的方式,还可以建立由包含  $p$  指向的页面的条目组成的表。这种邻接表的表现形式所占空间比最简单的表现形式减少一半。

下面将集中讨论从每一个页面发出的链接表,把这些技术应用到指向每个网页的链接也是一样的。为进一步减少表的存储空间,我们利用如下几点原则。

(1) 数组相似性:因为表中的许多行中有许多项是相同的,所以如果我们为几个相似行用一个原型行来代表,那么剩下的差异就可以根据原型行简洁地表现出来。

(2) 局域性:从一个网页发出的许多链接都是到邻近的网页。例如,在同一台主机上的网页,因此建议在编码目标链接的时候,使用小的整数来节约空间,可以使用差分编码的排序数组来实现这一点。不是直接存储每一条链接的终点网页的编号,而是存储与前一项的偏移量。

现在来实现这些压缩技术。先把每个 URL 作为一个字符串并对这些字符串排序。图 4-1 显示了排好序了的表的一部分。对于一个网页字典排序,URL 的域名部分应该被反转,因此 `www.lietu.com` 变为 `com.lietu.www`。这样至少能把相同前缀的字符串压缩掉,但是如果主要关心的是本地主机上的链接,就没有必要这样做。

```
1: www.lietu.com
2: www.lietu.com/demo
3: www.lietu.com/english
4: www.lietu.com/news
5: www.lietu.com/job
6: www.lietu.com/train
```

图 4-1 URL 字典排序

对每个 URL,取得它在排序后的 URL 地址列表中的位置作为表示它的唯一的整数。图 4-2 显示这样编号后的 Web 图的例子。在这个例子的序列中 `www.lietu.com/demo` 分配给整数 2,因为它在数组中排第二。



```

1: 1,2,4,8,16,32,64
2: 1,4,9,16,25,36,49,64
3: 1,2,3,5,8,13,21,34,55,89,144
4: 1,4,8,16,25,36,49,64

```

图 4-2 链接表的四行片断

每个网址分配一个唯一的长整数编号。

下面来分析 Web 图的相似性和局域性。许多网站都有模板，站点的每个网页链接到一个固定的网页集合，固定的网页集合常常包括：版权说明页、网站首页、站点地图页等。在这种情况下，Web 图中的相应网页的行有许多项是共同的。此外，按照 URL 字典排序，从一个网站出来的网页极有可能作为 Web 图中临近的行。

采取引用压缩（Reference compression）策略：从上往下遍历每一行，根据前 7 行来编码当前行。在图 4-2 的例子中，可以编码第 4 行和偏移量为 2 的行相同，只需要用 8 代替 9。仅使用前 7 行有两个有利条件。

（1）仅用 3 位来表示偏移量，这种选择在概率上是最优化的。

（2）固定的最大偏移量为一个像 7 这样小的值，避免了在许多可能表示当前行的候选原型中选择，因为这是代价昂贵的搜索。

有时候前 7 行没有一个能很好地表达当前行的原型，例如在每个不同站点的边界。这时候，简单的表示该行作为空行开始并向其中添加每一个整数。Web 图的每行中，使用差分编码（gap encoding）来存储增量，而不是实际值，基于值的分布来编码这些增量，可以获得更多的空间减少。这里提到的一系列技术使每个链接所需空间减少到 3 位，而不是最简单方法的 64 位。

虽然这些方法能够把 Web 图的表示存入内存中，但是仍然需要支持连通性查询的要求。如何查询一个网页指向哪些网页？首先，需要从哈希表中查询 URL 对应的行号。其次，需要跟踪偏移量指示的其他行来重建被压缩了的项，而且，这个间接过程可能重复多次。

然而，在实践中不会经常发生这样的事情。Web 图的构建过程中可以引入启发式控制：检查前 7 行哪个作为当前行的原型时，当前行和候选原型之间有个相似性的阈值，必须谨慎选择阈值。阈值如果设置得太高，会很少使用原型并重新表示许多行。如果设置得太低，许多行将

会用原型项表示，在查询时的重建速度就会变慢。

WebGraph (<http://webgraph.dsi.unimi.it/>) 是一个开源项目，它采用了引用压缩技术把图压缩成 BV 格式。

分析 BV 压缩格式的使用。因为 WebGraph 存储的节点是整型，不是直接的 URL 地址，首先需要把 Web 网页映射成 0 到  $n$  的节点编号，然后形成有向边的图，例如 example.arcs 文件中包括 (0,1,2,3,4) 5 个节点组成的图，文件的内容如下所示。

```
0 2
1 2
1 4
2 3
3 4
```

下面的命令将会产生一个压缩图到 bvexample 文件。

```
#java it.unimi.dsi.webgraph.BVGraph -g ArcListASCIIGraph example.arcs bvexample
```

生成压缩图以后可以统计图的出度。

```
#java it.unimi.dsi.webgraph.examples.OutdegreeStats -g BVGraph bvexample
```

执行结果如下所示。

```
The minimum outdegree is 0, attained by node 4
The maximum outdegree is 2, attained by node 1
The average outdegree is 1.0
```

读入的时候，数据是压缩格式存储的，直到需要时才解压缩相应的部分。

可以使用 birdeye (<http://code.google.com/p/birdeye/>) 绘制拓扑关系图，或者使用 <http://gephi.org/>。还可以使用 Neo4j 存储图数据。可以在 Java 中采用嵌入方式使用 Neo4j。一个 Neo4j 图数据库由以下几部分组成。

- 相互关联的节点。
- 有一定的关系存在。
- 在节点和关系上有一些属性。

## 4.4 本章小结

Java 代码中嵌入 SQL 语句的写法能够简化代码。

```
String empname;  
#sql { SELECT ename INTO :enpname FROM emp WHERE empno=28959 };
```

这样的写法比 JDBC 简单很多，但是直到现在 Eclipse 仍然不支持 SQLJ。Oracle 实现了一个 SQLJ 翻译器。

如果需要把抓取下来的数据存入数据库，则建议存入比 MySQL 稳定性更好的 PostgreSQL。



# 5

## 第 5 章

### 信息提取

---

搜索引擎经常要处理的文档格式包括 HTML、Word、PDF 等。这些文档格式中，有的文档是专有和非公开的格式，例如 Word 和 PDF；有的文档虽然是公开的标准，但是具体的实现却千差万别，例如 HTML。而且文档格式往往存在不同的版本，例如，Word 包括 doc 和 docx 两种格式，PDF 有从 1.0 到 1.7 及其扩展版等 9 种不同的格式，HTML 的版本 5 和版本 4 也不同。本章介绍如何从这样的文档中提取结构化信息。

#### 5.1 从文本提取信息

---

根据上下文判断当前字符串的含义，从而实现提取信息。为了提取网页中所有的类名，可以写一个简单的提取规则。

```
String rule = " class=\"<n>{classname}\""; //提取规则
TextExtractor ie = new TextExtractor(rule);
AdjList g = ie.getLattice(text);
g.extract("classname");
```

为了从具体的文本归类到抽象的类型，可以通过应用产生式到文本实现这点，每个产生式由左边的类型序列，和右边的类型序列组成。

若干个产生式组成文法。

## 5.2 从 HTML 文件中提取文本

从 HTML 文件中提取有效的文本，首先要判断网页编码，这样才能确保不出现乱码。其次从中提取需要的信息。因为很多网页中包括对用户不想要搜索的信息，例如广告、版权信息、导航条等，可以把网页保存成本地的文本文件，然后开发提取信息的程序，这样开发的时候就不用担心网速不稳定了。

有很多种方法可以实现网页信息提取。实现网页信息提取的方法一般可以分为两种类型：一种是针对特定的网页特征提取结构化信息；另一种是通用的信息提取方法，例如网页去噪。首先介绍字符集编码以及判断网页编码的基本过程，然后介绍网页信息提取所用到的一些工具和方法，例如 HTMLParser 和 NekoHTML。

### 5.2.1 字符集编码

ASCII 码（American Standard Code for Information Interchange，美国信息交换标准码）是目前计算机中用得最广泛的字符集及其编码，由美国国家标准局（ANSI）制定。它已被国际标准化组织（ISO）定为国际标准，称为 ISO 646 标准。ASCII 字符集由控制字符和图形字符组成。

在计算机的存储单元中，一个 ASCII 码值占一个字节（8 个二进制位），其最高位（b7）用作奇偶校验。所谓奇偶校验，是指在代码传送过程中用来检验是否出现错误的一种方法，一般分奇校验和偶校验两种。奇校验规定：正确的代码一个字节中 1 的个数必须是奇数，若非奇数，则在最高位 b7 添 1。偶校验规定：正确的代码一个字节中 1 的个数必须是偶数，若非偶数，则在最高位 b7 添 1。

ISO 8859，全称 ISO/IEC 8859，是国际标准化组织及国际电工委员会（IEC）联合制定的一系列 8 位字符集的标准，已经定义了 15 个字符集。

ASCII 码收录了空格及 94 个“可印刷字符”，足够英语使用。但是，其他使用拉丁字母的语言（主要是欧洲国家的语言），都有一定数量的变音字母，可以使用 ASCII 及控制字符以外的区域来储存及表示。

除了使用拉丁字母的语言外，使用西里尔字母的东欧语言、希腊语、泰语、现代阿拉伯语、希伯来语等，都可以使用这个形式来储存及表示。

- ISO 8859-1 (Latin-1) ——西欧语言。
- ISO 8859-2 (Latin-2) ——中欧语言。
- ISO 8859-3 (Latin-3) ——南欧语言。世界语也可用此字符集显示。
- ISO 8859-4 (Latin-4) ——北欧语言。
- ISO 8859-5 (Cyrillic) ——斯拉夫语言。
- ISO 8859-6 (Arabic) ——阿拉伯语。
- ISO 8859-7 (Greek) ——希腊语。
- ISO 8859-8 (Hebrew) ——希伯来语（视觉顺序）。
- ISO 8859-8-I ——希伯来语（逻辑顺序）。
- ISO 8859-9 (Latin-5 或 Turkish) ——它把 Latin-1 的冰岛语字母换走，加入土耳其语字母。
- ISO 8859-10 (Latin-6 或 Nordic) ——北日耳曼语支，用来代替 Latin-4。
- ISO 8859-11 (Thai) ——泰语，从泰国的 TIS620 标准字符集演化而来。
- ISO 8859-13 (Latin-7 或 Baltic Rim) ——波罗的语族。
- ISO 8859-14 (Latin-8 或 Celtic) ——凯尔特语族。
- ISO 8859-15 (Latin-9) ——西欧语言，加入 Latin-1 欠缺的法语及芬兰语重音字母，以及欧元符号。
- ISO 8859-16 (Latin-10) ——东南欧语言。主要供罗马尼亚语使用，并加入欧元符号。

很明显，ISO8859-1 编码表示的字符范围很窄，无法表示中文字符。但是，由于是单字节编码和计算机最基础的表示单位一致，所以很多时候仍旧使用 ISO8859-1 编码来表示，而且在很多协议上，默认使用该编码。

通用字符集(Universal Character Set, UCS)是由 ISO 制定的 ISO 10646(或称 ISO/IEC 10646)标准所定义的字符编码方式，采用 4 字节编码。

UCS 包含了已知语言的所有字符。除了拉丁语、希腊语、斯拉夫语、希伯来语、阿拉伯语、亚美尼亚语、格鲁吉亚语，UCS 还包括中文、日文、韩文等象形文字，以及大量的图形、印刷、数学、科学符号。



- UCS-2: 与 Unicode 的 2 字节编码基本一样。
- UCS-4: 4 字节编码, 目前是在 UCS-2 前加上 2 个全零的字节。

Unicode 是一种在计算机上使用的字符编码。它是 <http://www.unicode.org> 制定的编码机制, 要将全世界常用文字都囊括进去。Unicode 为每种语言中的每个字符设定了统一并且唯一的二进制编码, 以满足跨语言、跨平台进行文本转换、处理的要求。1990 年开始研发, 1994 年正式公布。随着计算机计算能力的增强, Unicode 得到了普及。

自从 Unicode 2.0 开始, Unicode 采用了与 ISO 10646-1 相同的字库和字码, ISO 也承诺 ISO 10646 将不会给超出 0x10FFFF 的 UCS-4 编码赋值, 使得两者保持一致。

Unicode 的编码方式与 ISO 10646 的通用字符集概念相对应, 目前的实际应用的 Unicode 版本对应于 UCS-2, 使用 16 位的编码空间。也就是每个字符占用 2 个字节, 基本满足各种语言的使用。实际上, 目前版本的 Unicode 尚未填满 16 位编码空间, 保留了大量空间作为特殊使用或将来扩展。

一个字符的 Unicode 编码是确定的, 但是在实际传输过程中, 由于不同系统平台的设计不一致, 以及出于节省空间的目的, 对 Unicode 编码的实现方式有所不同。Unicode 的实现方式称为 Unicode 转换格式 (Unicode Translation Format, UTF)。UTF 的两种实现方式如下所示。

- UTF-8: 8 位变长编码, 对于大多数常用字符集 (ASCII 中 0~127 字符) 它只使用单字节, 而对其他常用字符 (特别是中文、日文、韩文等象形文字), 它使用 3 字节。
- UTF-16: 16 位编码, 是变长码, 大致相当于 20 位编码, 值在 0 到 0x10FFFF 之间, 基本上就是 Unicode 编码的实现, 与 CPU 字序有关。

汉字编码有如下 4 种。

- GB2312 字集是简体字集, 全称为 GB2312 (80) 字集, 共包括国标简体汉字 6763 个。
- BIG5 字集是中国台湾地区的繁体字集, 共包括国标繁体汉字 13053 个。
- GBK 字集是简繁体字集, 包括了 GB 字集、BIG5 字集和一些符号, 共包括 21003 个字符。
- GB18030 是国家制定的一个强制性大字符集标准, 全称为 GB18030-2000, 它的推出使汉字集有了一个“大一统”的标准。

```
Set<String> charsetNames = Charset.availableCharsets().keySet();
System.out.println(charsetNames.contains("GB18030")); //返回 true
System.out.println(charsetNames.contains("GBK")); //返回 true
System.out.println(charsetNames.contains("GB2312")); //返回 true
```

在 Windows 系统中保存文本文件时通常可以选择编码为 ANSI、Unicode、Unicode big endian 和 UTF-8，这里的 ANSI 和 Unicode big endia 是什么编码呢？

使用 2 个字节来代表一个字符的各种汉字延伸编码方式，称为 ANSI 编码。在简体中文系统下，ANSI 编码代表 GB2312 编码；在日文操作系统下，ANSI 编码代表 JIS 编码。

UTF-8 以字节为编码单元，没有字节序的问题。UTF-16 以 2 个字节为编码单元，在解释一个 UTF-16 文本前，首先要弄清楚每个编码单元的字节序。

Unicode 规范中推荐的标记字节顺序的方法是 BOM (Byte Order Mark)。在 UCS 编码中有一个叫作“ZERO WIDTH NO-BREAK SPACE”的标记，它的编码是 FEFF。而 FFFE 在 UCS 中是不存在的字符，所以不应该出现在实际数据中。UCS 规范建议在传输字节流前，先传输标记“ZERO WIDTH NO-BREAK SPACE”。如果接收者收到 FEFF，就表明这个字节流是 Big-Endian 的；如果收到 FFFE，就表明这个字节流是 Little-Endian 的。因此标记“ZERO WIDTH NO-BREAK SPACE”又被称作 BOM。Windows 就是使用 BOM 来标记文本文件的编码方式的。

## 5.2.2 识别网页的编码

在实现从 Web 网页提取文本之前，首先要识别网页的编码，有时候还需要进一步识别网页所使用的语言。因为同一种编码可能对应多种语言，例如 UTF-8 编码可能对应英文或中文等任何语言。识别编码整体流程如下所示。

(1) 从 Web 服务器返回的 content type 头信息中提取编码，如果是 GB2312 类型的编码要当成 GBK 处理。

(2) 从网页的 Meta 标签中识别字符编码，如果和 content type 中的编码不一致，以 Meta 中声明的编码为准。

(3) 如果仍然无法确定网页所使用的字符集，需要从返回流的二进制格式判断。

(4) 往往采用统计的方法来估计网页的语言。JuniversalCharDet 就是采用统计的方法从二进制流识别字符编码。

Web 服务器返回的头信息中往往包含了网页编码。

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=UTF-8
Content-Length: 5367
Date: Sat, 21 May 2011 09:59:09 GMT
```

提取网页编码的代码如下所示。

```
//创建一个客户端，类似于打开一个浏览器
DefaultHttpClient httpClient = new DefaultHttpClient();

//创建一个 GET 方法，类似于在浏览器地址栏中输入一个地址
HttpGet httpget = new HttpGet("http://info.cm.hc360.com/");

//类似于在浏览器地址栏中输入回车，获得网页内容
HttpResponse response = httpClient.execute(httpget);

//查看返回的内容，类似于在浏览器查看网页源代码
HttpEntity entity = response.getEntity();

//取得头信息中声明的网页编码
String charSet = EntityUtils.getContentCharSet(entity);
System.out.println(charSet);
```

有的页面在头信息中不包括编码格式内容，需要从网页内部的 **meta** 标签中提取编码，如下所示。

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

下面利用正则表达式提取网页中的 **Meta** 信息。

```
Pattern pt = Pattern.compile("<meta(.*)charset=(.*)\\\"", Pattern.CASE_INSENSITIVE);
Matcher mc = pt.matcher(value); //匹配网页内容
if (mc.find()) {
    charSet = mc.group(2).trim();
}
```

另外一个和字符编码相关的问题是，有时候碰到 GB2312 编码的网页会有乱码问题，因为浏览器能正常显示包含 GBK 字符的 GB2312 编码网页，但是对于 Java 来说，需要用 GBK 字符集，所以要把设置字符集 GB2312 改成 GBK。

```
if(character_set!= null && character_set.toLowerCase().equals("gb2312")){
```



```
character_set = "GBK";
}
```

**JuniversalCharDet** (<http://code.google.com/p/juniversalchardet/>) 可以根据读入的字节流自动猜测页面或文件使用的字符集。实现原理是基于统计学的字符特征分析, 统计哪些字符是最常见的字符。**JuniversalCharDet** 可以检测的字符编码有: 中文、日文、韩文、西里尔文 (Cyrillic)、希腊文、希伯来文。

**UniversalDetector** 可以接收一个 **CharsetListener**。

```
UniversalDetector detector = new UniversalDetector(
    new CharsetListener() {
        public void report(String name) { //报告检测到的字符集
            System.out.println("charset = " + name);
        }
    }
);
```

调用 **JuniversalCharDet** 来自动判断网页的编码。

```
//取得网页内容二进制数组
byte[] bytes = EntityUtils.toByteArray(entity);

//构建一个org.mozilla.universalchardet.UniversalDetector对象的实例
UniversalDetector detector = new UniversalDetector(null);
//给编码检测器提供数据
detector.handleData(bytes, 0, bytes.length);
//通知编码检测器数据已经结束
detector.dataEnd();

//取得检测出的编码名
charSet = detector.getDetectedCharset();

//再次使用编码检测器之前, 先调用 UniversalDetector.reset()
detector.reset();
```

此外还有 **Jchardet** 和 **cpdetector**。**Jchardet** 是基于比较老的 **chardet** 模块, 而 **JuniversalCharDet** 基于新的 **UniversalCharDet** 模块, 因此检测结果更准确。

使用 **HttpClient** 下载网页, 完整的代码如下所示。

```
String charSet = EntityUtils.getContentCharSet(entity);
//如果头部没有说明字符集, 就需要查看页面源码
```

```

if (charSet == null || "".equals(charSet.trim())) {
    //http 头信息中无 charset
    //从 Meta 信息中找字符集
    //<meta http-equiv=Content-Type
    //content="text/html;charset=gb2312">
    //<meta http-equiv="content-type" content="text/html;
    //charset=UTF-8" />
    value = new String(bytes, defaultCharSet);
    //按默认编码转成字符串, 因为匹配中无中文, 所以字符串中可能的乱码对匹配没有影响
    Pattern pt = Pattern.compile("<meta(?:.*?) charset=(?:.*?)\\\"",
        Pattern.CASE_INSENSITIVE);
    Matcher mc = pt.matcher(value);
    if (mc.find()) {
        charSet = mc.group(2).trim();
    } else { // 否则是<?xml version="1.0" encoding="gb2312"?>格式的
        pt = Pattern.compile("<?xml(?:.*?) encoding=\\\"(?:.*?)\\\"",
            Pattern.CASE_INSENSITIVE);
        mc = pt.matcher(value);
        if (mc.find()) {
            charSet = mc.group(2).trim();
        }
    }
}

if (charSet != null && charSet.toLowerCase().equals("gb2312")) {
    charSet = "GBK";
}

```

### 5.2.3 网页编码转换为字符串编码

网页中的字符可能被转义成英文字符表示, 例如 “&#32593;&#31449;&#23548;&#33322;” 是“网站导航”的转义, 这类符号以“&”开始, 以“;”结束。http://commons.apache.org 项目中的 `StringEscapeUtils.unescapeHtml4(String str)` 方法可以把 HTML 编码的字符串转换成原文。下面的代码把输入转义后的字符串转换成原文。

```

String htmlStr = "&#32593;&#31449;&#23548;&#33322;";
String textStr = StringEscapeUtils.unescapeHtml4(htmlStr);
System.out.println(textStr); //输出: 网站导航

```

## 5.2.4 使用正则表达式提取数据

正则表达式可以定义一些与概率无关的提取模式。有的网站可以在线测试正则表达式，例如 <http://regexpal.com/>。

java.util.regex 包提供了对正则表达式的支持，其中 Pattern 类代表一个编译后的正则表达式。通过 Matcher 类根据给定的模式查找输入字符串，通过调用 Pattern 对象的 matcher 方法得到一个 Matcher 对象。使用正则表达式提取字符串的例子如下所示。

```
String example = "This is my small example string which I'm going to use for pattern matching.";
Pattern pattern = Pattern.compile("\\w+");
Matcher matcher = pattern.matcher(example);
//检查所有的出现
while (matcher.find()) {
    System.out.print("开始位置: " + matcher.start());
    System.out.print(" 结束位置: " + matcher.end() + " ");
    System.out.println(matcher.group());
}
```

用正则表达式检查 Email 的格式。\\w 与任何单词字符匹配，包括下划线。正则表达式如下所示。

```
\\w+@[\\w+\\.\\w+]+
```

正则表达式可以匹配 luogang@gmail.com 文本，可以在 <http://regexpal.com/> 在线测试正则表达式。

还需要 “.” 和 “-” 两个字符。例如，邮箱 zhangshna.Mr@163.com 在 “@” 符号之前还有个点 “.”。

检查 Email 的格式的语句。

```
String mailTo = "abc@sina.com.cn";
System.out.println(mailTo.matches( "[\\w[.-]]+@[\\w[.-]]+\\.([\\w]+)" )); //输出 true
```

正则表达式的原理是有限状态自动机，dk.brics.automaton 包含有限状态机的实现。BasicAutomata.makeChar 方法生成接收单个字符的自动机。

```
Automaton a = BasicAutomata.makeChar('@'); //创建一个字符 w 组成的自动机
```



`repeat` 方法重复多次, 例如重复字符 A 到 Z 至少一次。

```
Automaton a = BasicAutomata.makeCharRange('A', 'Z');
Automaton c = BasicOperations.repeat(a, 1); //指定最少重复次数
System.out.println(BasicOperations.run(c, "WW")); //输出 true
System.out.println(BasicOperations.run(c, "WWW")); //输出 true
```

`BasicOperations.concatenate` 方法连接两个自动机。

```
Automaton a = BasicAutomata.makeCharRange('A', 'Z');
Automaton b = BasicAutomata.makeChar('@');
Automaton c = BasicOperations.concatenate(a, b);
System.out.println(BasicOperations.run(c, "A@")); //输出 true
System.out.println(BasicOperations.run(c, "AW")); //输出 false
```

匹配数字。

```
Automaton num = Automaton.minimize((new RegExp("[0-9]+")).toAutomaton());
System.out.println(BasicOperations.run(num, "12356756700")); //输出 true
```

很多时候对匹配对象有更多的要求。根据上下文过滤匹配结果要用到环视结构。如果条件位于要提取信息的后面, 则叫作向前看表达式, 否则叫作向回看。

一个向回看的表达式从模式开始, 直到向回看的表达式结束为止。

(?<=X) X, 按条件 X 向回寻找

例如, 查找 “http://” 后面的文本写作: (?<=http://)。

```
//查找 “http://” 后面的文本
Pattern pat = Pattern.compile( "(?<=http://)\\S+" );

String str = "The Java2s website can be found at http://www.java2s.com. There, you can find some Java examples.";

Matcher matcher = pat.matcher(str);
while (matcher.find())
    System.out.println(":" + matcher.group() + ":");
```

提取网页中的链接。

```
String pageContents = "<a href=\"http://www.lietu.com\">猎兔</a>";
Pattern p = Pattern.compile("<a\\s+href\\s*=\\s*\"?(.*?)\"? [\\\"|>]",
    Pattern.CASE_INSENSITIVE); //忽略大小写
```

```

Matcher m = p.matcher(pageContents);
while (m.find()) { //打印网页中所有链接
    String link = m.group(1).trim();
    System.out.println(link);
}

```

有些链接的形式如下所示。

```
<a href='http://www.lietu.com'>猎兔</a>
```

为了更好的匹配单引号，可以把模式修改成如下所示。

```
"<a\\s+href\\s*=\\s*["'"]?(.*?)[\\"'>"
```

例如，“2009-12-6”的格式用正则表达式匹配如下所示。

```

Pattern p = Pattern.compile("\\d{2,4}-\\d{1,2}-\\d{1,2}");
Matcher m = p.matcher(inputStr);
if(m.find()){
    String strDate = m.group();
}

```

其他的一些匹配日期的正则表达式有：“\\d{2,4}\\d{1,2}\\d{1,2}” “\\d{2,4}年\\d{1,2}月\\d{1,2}日”，以及\\d{2,4}\\d{1,2}\\d{2,4}。

图 5-1 是一个包含地址信息的网页。利用 HTMLParser 解析包可以方便地解析网页，提取想要的网页信息，但是在有些情况下利用正则表达式可能更方便（如果对正则表达式不熟悉，那么请参考相关 Java 正则表达式教程）。

邮政编码查询					
邮编查询:	<input type="text"/>	<input type="button" value="查询"/>	区号查询:	<input type="text"/>	<input type="button" value="查询"/>
省名查询:	<input type="text" value="山西"/>	<input type="button" value="查询"/>	市名查询:	<input type="text"/>	<input type="button" value="查询"/>
县名查询:	<input type="text"/>	<input type="button" value="查询"/>	村名查询:	<input type="text"/>	<input type="button" value="查询"/>

省名	地区	县市	乡镇村	邮政编码	区号
山西	朔州	怀仁县	鹅毛口	038301	0349
山西	朔州	怀仁县	海北头	038301	0349
山西	朔州	怀仁县	何家堡	038301	0349
山西	朔州	怀仁县	河头	038301	0349
山西	朔州	怀仁县	金沙滩	038302	0349
山西	朔州	怀仁县	里八庄	038301	0349

图 5-1 地址页面

首先,根据要提取的网页信息查看网页源代码,这里提取的是网页表格中地名相关的信息,查看网页源码如下所示。

```
</table> <br><table width="533" border="0" cellpadding="0" cellspacing="1" align="center"
bgcolor="#3366CC"><tr align="center" bgcolor="#6699CC"><td width="70">省名</td><td
width="100">地区</td><td width="100">县市</td><td width="50">乡镇村</td><td width="60">邮政
编码</td><td width="50">区号</td></tr><tr align="center"
onmouseout="this.style.background='#FFFFFF'"
onmouseover="this.style.background='BDDFFF'" bgcolor="#FFFFFF"><td><a
href=Code_Default.asp?Type=Sm&SearchKeyStr=山西>山西</a></td><td><a
href=Code_Default.asp?Type=Ds&SearchKeyStr=朔州>朔州</a></td><td><a
href=Code_Default.asp?Type=Xs&SearchKeyStr=怀仁县>怀仁县</a></td><td><a
href=Code_Default.asp?Type=Ys&SearchKeyStr=鹅毛口>鹅毛口</a></td><td><a
href=Code_Default.asp?Type=Yb&SearchKeyStr=038301>038301</a></td><td><a
href=Code_Default.asp?Type=Qh&SearchKeyStr=0349>0349</a></td></tr><tr align="center"
```

根据要提取的表格信息构造正则表达式如下所示。

```
<td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td></tr>"
```

在这里要注意“(.\*?)”用法,括号表示分组,即每一个数据项表示一组,“?”表示非贪婪匹配,仅匹配到紧跟其后的“</a>”为止。

其次,根据读取的网页字符流和正则表达式提取表格信息,其相关代码如下所示。

```
//输入参数 input 是当前网页的内容字符串
public static void parserHtml(String input) {
    String regex =
"<td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td><td><a.+?>(.*?)</a></td></tr>";
    Pattern p = Pattern.compile(regex); //编译正则表达式
    Matcher m = p.matcher(input); //匹配模式
    while (m.find()) { //按行提取表格数据
        String province = m.group(1);
        String area = m.group(2);
        String city = m.group(3);
        String village = m.group(4);
        String postcode = m.group(5);
        String code = m.group(6);
        System.out.println(province); //打印出山西
        System.out.println(area); //打印出朔州
```



```

        System.out.println(city); //打印出怀仁县
        System.out.println(villag); //打印出鹅毛口
        System.out.println(postcode); //打印出 038301
        System.out.println(code); //打印出 0349
    }
}

```

默认情况下是按行匹配。`Pattern.MULTILINE` 启用多行模式,通过 `Pattern` 类的静态方法 `static Pattern compile(String regex, int flags)` 来设定标志。

多行模式影响 “^” 和 “\$” 的匹配。例如, 如下的模式会找到行开始位置的数字。

```

Pattern p = Pattern.compile("^([0-9]+).*", Pattern.MULTILINE);
Matcher m = p.matcher(...);
while (m.find()) {
    String number = m.group(1);
    ...
}

```

单行模式影响 “.” 的匹配。

开启全局匹配, 匹配所有成功匹配项。

## 5.2.5 结构化信息提取

一般把要提取的数据结构定义成为一个类, 然后用一个解析网页的方法根据输入网页返回解析出来的类实例。例如, 需要实现一个招聘职位搜索引擎, 从招聘网页提取职位信息。

首先, 定义好用来接收网页数据的 `POJO` (Plain Old Java Object) 类 `Job`, 用来描述职位信息。

```

public class Job {
    public String comName = null;
    public String positionName = null;
    public String email = null;
    public String releaseData = null;
    public String city = null;
    public String number = null;
    public String experience = null;
    public String salary = "面议";
}

```

```

public String knowledge = null;
public String acceptResumeLanguage = null;
public String positionDescribe = null;
public String comintro = null;
public String comHomePage = null;
public String comAddress = null;
public String postCode = null;
public String fax = null;
public String connectPerson = null;
public String telephone = null;
public String languageAbility = null;
public String funtype_big = null;
public String funtype = null;
public String province = null;
public String toString(){
    return "公司名称: "+comName+"\n 职位名称: "+positionName+
        "\n 电子邮箱: " +email+"\n 发布日期: "+releaseData+
        "\n 工作地点: "+city+"\n 招聘人数: "+number+
        "\n 工作年限: "+experience+"\n 薪水范围: "+salary+
        "\n 学历: "+knowledge+
        "\n 接受简历语言: "+acceptResumeLanguage+
        "\n 职位描述: "+positionDescribe+"\n 公司简介: "+comintro+
        "\n 公司网站: "+comHomePage+"\n 地址: "+comAddress+
        "\n 邮政编码: "+postCode+"\n 传真: "+fax+
        "\n 联系人: "+connectPerson+"\n 电话: "+telephone+
        "\n 外语要求: "+languageAbility;
}

```

然后，从发布这个职位的网页中提取公司名称，查看公司名称周围的特征。

```
<td align="left" class="title02">北京亿达网通科技发展有限公司</TD>
```

可以利用 AndFilter 来处理。

//提取公司名称的 Filter

```

NodeFilter filter_title = new AndFilter(new TagNameFilter("TD"),
    new HasAttributeFilter("class", "title02"));

```

提取公司名称。

```

NodeList nodelist = parser.extractAllNodesThatMatch(filter_title);
Node node_title = nodelist.elementAt(0);
String comName = extractText(node_title.toHtml());

```

```
comName = comName.replaceAll("[\t\n\f\r ]+", ""); //去掉多余的空格
```

从职位详细页面 URL 地址提取职位信息的完整的过程如下所示。

//输入网址, 返回和该网址正文信息对应的职位对象

```
public Job extractContent(String url){
    Job position = new Job();
    Parser parser = new Parser(url);
    //设置编码方式
    parser.setEncoding("GB2312");
    //提取公司名字的 Filter
    NodeFilter filter_title = new AndFilter(new TagNameFilter("TD"),
        new HasAttributeFilter("class", "title02"));
    //提取公司名称
    NodeList nodelist = parser.extractAllNodesThatMatch(filter_title);
    Node node_title = nodelist.elementAt(0);
    position.comName = extractText(node_title.toHtml());
    position.comName = position.comName.replaceAll("[\t\n\f\r ]+", "");
    //提取职位名称的 Filter
    NodeFilter filter_job_name = new AndFilter(new TagNameFilter("td"),
        new HasAttributeFilter("bgcolor", "#FFEE00"));
    NodeFilter job_description_end1 = new AndFilter(new TagNameFilter("table"),
        new HasAttributeFilter("width", "100%"));
    NodeFilter job_description_end2 = new HasAttributeFilter("cellpadding", "5");
    NodeFilter company_description = new AndFilter(new TagNameFilter("table"),
        new HasAttributeFilter("width", "98%"));
    //提取职位的名称
    parser.reset();
    nodelist.removeAll();
    nodelist = parser.extractAllNodesThatMatch(filter_job_name);
    Node node_job_name = nodelist.elementAt(0);
    position.positionName = extractText(node_job_name.toHtml());
    //去掉空格
    position.positionName =
        position.positionName.toString().replaceAll("[\t\n\f\r ]+", "");
    //提取职位描述
    NodeFilter job_description_end =
        new AndFilter(job_description_end1, job_description_end2);
    parser.reset();
    NodeList nodelist_description =
        parser.extractAllNodesThatMatch(job_description_end);
    Node node_job_description = nodelist_description.elementAt(0);
```



```

position.positionDescribe = extractText(node_job_description.toHtml());
position.positionDescribe =
    position.positionDescribe.replaceAll("[\t\n\f\r ]+", " ");
//提取公司简介
parser.reset();
odelist_description.removeAll();
odelist_description = parser.extractAllNodesThatMatch(company_description);
Node node_company_description = oodelist_description.elementAt(0);
//处理公司简介
position.comintro = doCompanyDescription(node_company_description);
return position;
}

```

## 5.2.6 表格

内容页不一定是标题和正文两列。有些内容页的有效信息在表格中,例如,从 [http://eservice.beijing.gov.cn/sj/xzfwzy/fwsx/201209/t20120912\\_105347.htm](http://eservice.beijing.gov.cn/sj/xzfwzy/fwsx/201209/t20120912_105347.htm) 中提取“办理条件、申报材料、办理流程”等信息,可以把从表格中提取的有效数据组织成键/值对形式。

```

public static HashMap<String, String> extract(HashSet<String> keys,String url) {
    HashMap<String,String> data = new HashMap<String,String>(); //提取出来的数据

    String htmlContent = HttpUtil.getHtml(new URL(url));
    Document doc = Jsoup.parse(htmlContent);

    Elements tables = doc.getElementsByTag("table");
    for(Element table:tables){
        Elements rows = table.getElementsByTag("tr");
        for (Element row : rows) {
            Elements rowItems = row.select("td");
            if(rowItems.size()>1){
                String key = rowItems.get(0).text(); //取得键
                if(!keys.contains(key)){ //如果对这个键不感兴趣,则忽略它
                    continue;
                }
                String value = rowItems.get(1).text(); //从第二个td标签中取得值
                if(!StringUtils.isEmpty(value)){
                    data.put(key, value); //填入键/值对
                }
            }
        }
    }
}

```

```

    }
  }
}
return data;
}

```

把这里的“键/值”对叫作“属性/值”对。

### 5.2.7 网页的 DOM 结构

虽然表示网页的 HTML 文档格式不如 XML 规范，但是仍然可以把它转换成 DOM（文档对象模型）树组织的结构。其中标签是 DOM 树内部的节点，而详细的文本、图像或者超链接则是叶节点。图 5-2 展示了 HTML 的一部分以及它相应的 DOM 树。

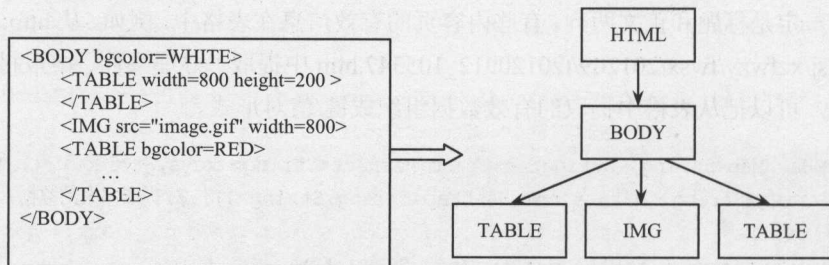


图 5-2 HTML 的一部分以及它相应的 DOM 树

在 Firefox 浏览器中，使用 DOM 查看器（DOM Inspector）和 Firebug 两个插件工具都可以看到网页的 DOM 树。

访问 <http://www.lietu.com/>。在浏览器菜单中，选择“工具”→“DOM 查看器”，打开 DOM 查看器。在 DOM 查看器的左侧视图中，会看到 DOM 节点的树状图。

如果觉得依次展开 DOM 节点中每层很不方便，可以使用 Inspect Element 扩展，它能迅速找到 DOM 查看器中的指定元素。XPath Helper (<http://xpathhelper.mozdev.org/>) 可以扩展 DOM 查看器，它在 DOM 查看器上增加了一个可以输入 XPath 的工具条，可以在当前网页计算 XPath 表达式。

通过 Firebug，可以在 Firefox 中查看任何页面已解析的文档对象模型，可以获得每个 HTML

元素、属性和文本节点的详细信息；可以看到每个页面样式表中的所有 CSS 规则；也可以看到每个对象的所有脚本属性。在 Firefox 中安装 Firebug 插件后，输入网址 “http://www.lietu.com” 即可看到网页 DOM 树。比较而言，DOM 查看器中显示的网页 DOM 树概念更符合网页解析器 NekoHTML 所生成的 DOM 树。

org.w3c.dom 是操作 DOM 的标准接口，其中 DOM 树中的节点叫作 Node。整个 HTML 文档叫作 Document，也就是 DOM 树中的根节点，所以 Document 是 Node 的子类。Attr 表示节点中的属性，例如，IMG 类型的节点包括 src 属性。Comment 表示一个注释类型的节点，所以它也是 Node 的子类。

## 5.2.8 使用 Jsoup 提取信息

Jsoup (<http://jsoup.org/>) 和 NekoHTML 一样，能够根据不规范的 HTML 格式生成 DOM 树，补全有开始没结束的标签。例如，把 `<p>Lorem <p>Ipsum` 解析成为 `<p>Lorem</p> <p>Ipsum</p>`，这里的两个 `</p>` 是等价标签。

NekoHTML 需要 Xerces，而 Jsoup 没有依赖其他的 jar 包，只需要 jsoup-1.6.3.jar 这一个 jar 包。在爬虫项目中增加对 jsoup-1.6.3.jar 的引用。

Jsoup.connect(String url)方法创建一个新的连接。get()方法取得并解析一个 HTML 文件。如果发生错误，就抛出一个 IOException，从一个 URL 解析 HTML 的代码。

```
String url = "http://www.lietu.com";
Document doc = Jsoup.connect(url).get(); //解析的结果就是一个文档对象
```

在用 Jsoup 获取网络数据时出现如下错误。

```
org.jsoup.UnsupportedMimeTypeException: Unhandled content type. Must be text/*,
application/xml, or application/xhtml+xml.
Mimetype=application/x-javascript, URL
```

忽略它的返回类型。

解决方案如下所示。

```
String script = Jsoup.connect(url).ignoreContentType(true).execute().body();
```

返回的 Document 对象中包含了 Elements 和 TextNodes。当然 Document 也可以认为是一个



特殊的 Element，因为 Document 本身就是继承自 Element。

设置连接参数。

```
Document doc = Jsoup.connect("http://www.lietu.com/")
    .data("query", "Java") //请求参数
    .userAgent("jsoup") //设置 User-Agent
    .Cookie("auth", "token") //设置 Cookie
    .timeout(3000) //设置连接超时时间
    .post(); //使用 POST 方法访问 URL
```

在 Jsoup 中得到并处理响应状态码。

```
Connection.Response response = Jsoup.connect(sitemapPath)
    .userAgent("Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.21 (KHTML,
like Gecko) Chrome/19.0.1042.0 Safari/535.21")
    .timeout(10000)
    .execute();

int statusCode = response.statusCode();
if(statusCode == 200) {
    Document doc = connection.get();
    Elements element = doc.select("loc");
    for (Element urls : element) {
        System.out.println(urls.text());
    }
} else {
    System.out.println("received error code : " + statusCode);
}
```

也可以直接从网页内容字符串得到文档对象。

```
String html = "<html><head><title>First parse</title></head>"
    + "<body><p>Parsed HTML into a doc.</p></body></html>";
Document doc = Jsoup.parse(html);
```

如果本地硬盘中的一个文件缓存了 HTML 页面，可以加载这个文件并解析内容。

```
File input = new File("d:/tmp/input.html");
Document doc = Jsoup.parse(input, "UTF-8", "http://www.lietu.com");
```

Jsoup.connect 下载网页没有多次重试功能，可以使用 HttpClient 下载网页后把字符串传给 Jsoup 解析。

```
//HttpClient 得到 HTML 内容
Document doc = Jsoup.parse(html);
```

得到网页标题。

```
Document doc = Jsoup.connect("http://www.lietu.com/").get();
String title = doc.title(); //取出网页的标题
```

Element.text()方法输出节点对应的文本。例如，对于 HTML 文本：<p>Hello <b>there</b> now!</p>，调用 p.text()方法，返回"Hello there now!"。

Element.html()返回节点代表的整个 HTML，会保留换行符。Element.outerHtml()则会返回包含描述 Element 标签在内的所有 HTML 内容。

Jsoup 支持使用 DOM 来查找、取出数据，也可以使用 CSS 选择器来查找、取出数据。例如选出带有 href 属性的 a 类型的标签。

```
Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
```

可以提取下面这个 a 标签中的网址。

```
<a href="/www/detail/detail.jsp?DOCID=16951" target="_blank" title="企业资产损失所得税税前扣除相关表单">
```

提取 a 标签中的 title 属性包括了“讲话”字样。

```
Elements links = doc.select("a[title~=讲话]");
```

提取网页中的链接的完整例子。

```
String url = "http://www.lietu.com";
Document doc = Jsoup.connect(url).get();
Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
    String linkText = link.text(); //得到锚点上的文字说明
    System.out.println(linkHref+" "+linkText); //输出 URL 地址和锚点上的文字说明
}
```

除了 CSS 选择器，Jsoup 还提供了类似于 JQuery 的操作方法来取出和操作数据。getElementById 和 getElementsByTag 方法跟 JavaScript 中的方法名称是一样的，功能也完全一致，可以根据节点名称或者是 HTML 元素的 id 来获取对应的元素或者元素列表。

```
File input = new File("D:/test.html");
Document doc = Jsoup.parse(input, "UTF-8", "http://www.lietu.com/");

Element content = doc.getElementById("content"); //通过 id 名称获取对应的元素
Elements links = content.getElementsByTag("a"); //通过类型获取元素列表
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值, 也就是 URL 地址
    String linkText = link.text(); //得到锚点上的文字说明
}
```

可操作 HTML 元素、属性、文本。

```
for (Element link : links) {
    System.out.println(" * a: <%s> (%s)", link.attr("abs:href"), link.text());
}
```

按照特定标签类型筛选, 格式是: `tagName.className`, 可以通过 `class` 或者 `id` 信息选取元素。例如, 选择商品名称所在的 Div 区域`<div class=p-name>`。

```
Element content = document.select("div.p-name").first(); //通过 class 选取元素
```

想要选择`<div id="content">`, 可以使用 CSS ID 选择器采用格式`"#id"`。

```
Document document = Jsoup.connect("http://www.qualcomm.com/innovation").get();
Element content = document.select("#content").first();
System.out.println(content.html());
```

"`<!-- -->`"是网页中的注释标签。注释是节点, 用节点名`#comment`标识, 去掉注释节点的代码。

```
public class RemoveComments {
    public static void main(String... args) {
        String h = "<html><head></head><body>" +
            "<div><!-- foo --><p>bar<!-- baz --></div><!-- qux--></body></html>";
        Document doc = Jsoup.parse(h);
        removeComments(doc);
        System.out.println(doc.html());
    }

    private static void removeComments(Node node) {
        for (int i = 0; i < node.childNodes().size(); i++) {
            Node child = node.childNodes(i);
            if (child.nodeName().equals("#comment"))
```



```

        child.remove();
    else {
        removeComments(child);
        i++;
    }
}
}
}
}
}

```

每一个节点在 DOM 树中都有一个特定的位置。Node 接口中有一些发现一个指定节点的周边节点的方法。

Jsoup 提供了图形化方式导航：parent()得到父亲节点、children()得到所有的孩子节点、child(int index)得到指定的孩子节点。

```

string html = "<div id='demo'><span style='color:red;'><h1>Hello World!</h1></span><div id='innerDiv'>inner div</div></div>";

```

FirstChild 属性返回所有子节点的第一个节点，FirstChild 返回 “<span style='color:red;'><h1>Hello World!</h1></span>” 节点。

LastChild 属性返回所有子节点的最后一个节点，LastChild 返回 “<div id='innerDiv'>inner div</div>” 节点。

children()返回当前节点所有直接一代的子节点的集合，不包括跨代子节点，这里返回“<span style='color:red;'><h1>Hello World!</h1></span>” 和 “<div id='innerDiv'>inner div</div>” 两个节点。

NodeVisitor 接口用来遍历 DOM 树中的节点。这个接口提供了两个方法，一个叫作 head，另外一个叫作 tail。当第一次看到节点时，调用 head 方法。当这个节点所有的孩子都已经访问过以后，调用 tail 方法。例如，可以使用 head 创建一个节点的开始标签，用 tail 创建结束标签。

```

Document doc = Jsoup.parse(html);

//用一个NodeVisitor 对象构造一个NodeTraversor
NodeTraversor nd = new NodeTraversor(new NodeVisitor() {

    @Override
    public void tail(Node node, int depth) {
        //处理代码
    }
});

```

```

    }

    @Override
    public void head(Node node, int depth) {
    }
});

nd.traverse(doc.body());

```

通过 Collector 收集符合条件的元素节点。

调用 Evaluator 的 matches 方法判断元素节点是否符合条件。

Element.absUrl()方法得到 URL 绝对地址。

```
String url = dl.select("a").absUrl("href");
```

输出网页中的 URL 绝对地址的完整代码如下所示。

```

Elements links = doc.select("a[href]"); //过滤带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.absUrl("href"); //得到 URL 绝对地址
    System.out.println(linkHref);
}

```

这个方法在 Jsoup 1.7.3 版本中有 bug，还不能使用，因此可以使用 UrlResolver 类中的 resolveUrl 方法实现提取 URL 绝对地址。

```

Elements links = doc.select("a[href]"); //过滤带有 href 属性的 a 标签
for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址，但是有可能有相对 URL
    linkHref = UrlResolver.resolveUrl(baseUrl, linkHref); //得到 URL 绝对地址
    System.out.println(linkHref);
}

```

Node.replaceWith(...)可以替换 DOM 中的一个节点。

可以用 HttpClient 实现下载网页，而用 Jsoup 解析网页。集成 Jsoup 和 HttpClient 的例子如下所示。

```

String url = "http://ircs.p5w.net/ircs/gszz/newsList.do";
String content = HttpUtil.getContent(url);

```

```
Document doc = Jsoup.parse(content);

Elements links = doc.select("a[href]");
for (Element link : links) { //遍历每个链接, 集合里面的每一个元素写在前面
    String title = link.text();
    System.out.println(title);
}
```

## 5.2.9 使用 XPath 提取信息

NekoHTML 可以把 HTML 文档转换成 XML 文档。XML 文档的某一部分可以用 XPath 来描述, 可以用计算节点特征的方法找到覆盖主要正文的内容节点。找到这样一个节点后, 可以把这个节点用 XPath 表示出来。

```
/HTML[1]/BODY[1]/DIV[1]/TABLE[1]/TBODY[1]/TR[1]/TD[1]/TABLE[1]/TBODY[1]/TR[1]/TD[1]
```

当 DOM 树的节点有删除或修改后, XPath 的绝对路径表示方式就失效了。还可以用 XPath 中的相对路径来选择节点, 例如, 选择网页中所有链接的 XPath 是 “//a”。选择一个标题 div 可能是 “//div[@class='title']”。如果路径以单斜线 (/) 开始, 那么该路径就表示到一个元素的绝对路径。如果路径以双斜线 (//) 开头, 那么该路径就表示相对路径, 即选择文档中所有满足双斜线之后规则的元素。

星号 (\*) 表示选择所有由星号之前的路径所定位的元素, 例如, “/HTML[1]/BODY[1]/\*”。

使用 Firebug 可以自动生成 XPath。Firebug 生成的 XPath 中如果含有 TBODY 标签, 需要把 TBODY 去掉, 否则不能正确获取网页的内容。如果 Firebug 生成的 XPath 为 /html/body/table/tbody/tr, 那么则需要修改为 /html/body/table/tr。

XPath 的 Tag 必须大写。

```
String divXpath = "//DIV";//正确
String divXpath = "//div";//错误
```

节点的属性按照页面中实际的大小写进行书写。

```
//DIV[@class="title"]/EM[@class="right"]/A
```

找到页面中 DIV 的 class 为 title 的大块, 然后向下找节点为 EM, 同时其 class 为 right 的小块, 再继续延伸找到此小块下的 A 节点。



为了支持通过 XPath 取得节点, Java 项目必须导入 xalan.jar。下面的例子是提取当当网图书的 ISBN 信息。

```
DOMParser parser = new DOMParser();

parser.setProperty("http://cyberneko.org/html/properties/default-encoding", "gb2312");
parser.setFeature("http://xml.org/sax/features/namespace", false);
String bookURL =
    "http://product.dangdang.com/product.aspx?product_id=20733895";
parser.parse(bookURL);

Document doc = parser.getDocument();
String isbnXPath = "/HTML/BODY/DIV[3]/DIV[6]/DIV[2]/DIV/DIV[3]/UL/LI[9]";
NodeList products;
products = XPathAPI.selectNodeList(doc, isbnXPath);
System.out.println("found: " + products.getLength());
Node node = null;
for (int i = 0; i < products.getLength(); i++) {
    node = products.item(i);
    System.out.println(i + ":\n" + node.getTextContent());
}
```

通过相对路径提取淘宝网商品中的价格。

```
String goodUrl = "http://item.taobao.com/item.htm?id=12829488341";

HtmlPage page = webClient.getPage(goodUrl);
HtmlElement element = page.getFirstByXPath("//*[@id=\"J_StrPrice\"]"); //通过 id 选取
```

通过类别选取商品的 URL 地址。

```
String searchURL = "http://search.taobao.com/search?q=item+htm+id";

WebClient webClient = new WebClient();
webClient.setJavaScriptEnabled(false);
HtmlPage page = webClient.getPage(searchURL);

List<HtmlElement> elements =
    (List<HtmlElement>)
    page.getByXPath("//a[@class='EventCanSelect']");

ArrayList<String> urls = new ArrayList<String>();
for (HtmlElement e : elements) {
```

```
String title = e.getAttribute("title");
int pos = title.indexOf("http");
urls.add(title.substring(pos));
}
```

Jsoup 本身不支持 Xpath, <https://github.com/code4craft/xsoup> 让 Jsoup 支持 XPath 选择器, 使用 xsoup。

```
String html = "<html><div><a href='https://github.com'>github.com</a></div>" +
    "<table><tr><td>a</td><td>b</td></tr></table></html>";

Document document = Jsoup.parse(html);

String result = Xsoup.compile("//a/@href").evaluate(document).get();
Assert.assertEquals("https://github.com", result);

List<String> list = Xsoup.compile("//tr/td/text()").evaluate(document).list();
Assert.assertEquals("a", list.get(0));
Assert.assertEquals("b", list.get(1));
```

### 5.2.10 HTMLUnit 提取数据

HTMLUnit 提取数据, 得到动态网页内容。

```
final WebClient webClient = new WebClient(BrowserVersion.FIREFOX_3_6);
webClient.getOptions().setTimeout(20000);
webClient.getOptions().setJavaScriptEnabled(true);
webClient.getOptions().setThrowExceptionOnScriptError(false);

String theContent1 =
    webClient.getPage(theURL).getWebResponse().getContentAsString();
```

“<a href="javascript:next(0)" title="下一页">下一页</a>”就是一个动态锚点, 需要得到它的真实地址才能实现翻页抓取。根据锚文本得到两种动态锚点如下所示。

```
HtmlAnchor myAnchor = page.getAnchorByText("下一页");
HtmlAnchor myAnchor = page.getAnchorByHref("javascript:next(0)");
```

自动单击得到的动态锚点。

```
final HtmlPage newPage = myAnchor.click();
```

得到 `HtmlPage` 对象的 URI 地址。

```
String url =
    newPage.executeJavaScript("document.location").getJavaScriptResult().toString();
```

HTMLUnit 底层使用 Rhino 解释 JavaScript, 使用 HttpClient 处理 Cookie。

### 5.2.11 网页结构相似度计算

自动提取结构化信息的关键是从同样类型的实例中发现编码模版。为了确定两个网页是否由同一个网页模板生成, 需要计算两个网页的结构相似度。一个方法是从 HTML 编码字符串检测重复的模式。检测的方法有最长公共子序列 (Longest Common Subsequence, LCS) 和树编辑距离两种。

举例说明, 两个序列  $s_1$  和  $s_2$  的最长公共子序列。 $s_1 = \{a, b, c, b, d, a, b\}$ ,  $s_2 = \{b, d, c, a, b, a\}$ , 从前往后找,  $s_1$  和  $s_2$  的最长公共子序列为  $LCS(s_1, s_2) = \{b, c, b, a\}$ , 如图 5-3 所示。

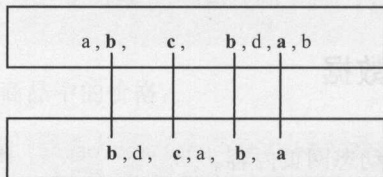


图 5-3 最长公共子序列

LCS 问题的最优解只取决于其子序列 LCS 问题的最优解, 而非最优解对问题的求解没有影响。同时子序列 LCS 的值和当前对比字符的值一旦确定, 则此后过程的 LCS 计算不再受此前各状态及决策的影响。因为满足动态规划的最优化原理和无后效性原则, 因此使用动态规划的思想计算 LCS 的方法: 引进一个二维数组  $num[i][j]$ , 用  $num[i][j]$  记录  $s_1$  的前  $i$  个长度的子串与  $s_2$  的前  $j$  个长度的子串的 LCS 的长度。

自底向上进行递推计算, 那么在计算  $num[i][j]$  之前,  $num[i-1][j-1]$ ,  $num[i-1][j]$  与  $num[i][j-1]$  均已计算出来。此时再根据  $s_1[i-1]$  和  $s_2[j-1]$  是否相等, 就可以计算出  $num[i][j]$ 。

采用动态规划的方法计算两个序列的最长公共子序列的实现代码如下所示。

```
public static <E> List<E> longestCommonSubsequence(E[] s1, E[] s2){
    int[][] num = new int[s1.length+1][s2.length+1]; //二维数组
```



```

//实际算法
for (int i = 1; i <= s1.length; i++)
    for (int j = 1; j <= s2.length; j++)
        if (s1[i-1].equals(s2[j-1]))
            num[i][j] = 1 + num[i-1][j-1];
        else
            num[i][j] = Math.max(num[i-1][j], num[i][j-1]);

System.out.println("length of LCS = " + num[s1.length][s2.length]);

int s1position = s1.length, s2position = s2.length;
List<E> result = new LinkedList<E>();

while (s1position != 0 && s2position != 0) {
    if (s1[s1position - 1].equals(s2[s2position - 1])) {
        result.add(s1[s1position - 1]);
        s1position--;
        s2position--;
    }
    else if
        (num[s1position][s2position - 1] >= num[s1position - 1][s2position]) {
        s2position--;
    }
    else {
        s1position--;
    }
}
Collections.reverse(result);
return result;
}

```

比较网页结构相似度的基本流程是：首先把网页抽象成一个 Node 数组，然后比较两个 Node 数组的最长公共子序列。输入两个 URL 地址，返回网页相似度的实现代码如下所示。

```

public static double getPageDistance(String urlStr1, String urlStr2) {
    ArrayList<Node> pageNodes1 = new ArrayList<Node>(); //网页转换成 Node 数组

    URL url = new URL(urlStr1);
    Node node;
    Lexer lexer = new Lexer (url.openConnection ());

```

```

lexer.setNodeFactory(new PrototypicalNodeFactory ());
while (null != (node = lexer.nextNode ())) {
    pageNodes1.add(node);
}

ArrayList<Node> pageNodes2 = new ArrayList<Node>();
URL url2 = new URL(urlStr2);

lexer = new Lexer (url2.openConnection ());
lexer.setNodeFactory(new PrototypicalNodeFactory ());
while (null != (node = lexer.nextNode ())) {
    pageNodes2.add(node);
}

int lcs = longestCommonSubsequence (pageNodes1, pageNodes2);
return lcs / (double)Math.min(pageNodes1.size(), pageNodes2.size()) ;
}

```

## 5.2.12 提取标题

a 标签中描述链接网页的文字叫作锚文本。

```
<a href="URL 链接">锚文本</a>
```

有 3 种提取标题的方式。

- 目录页上的锚点描述文字。
- 详细页本身的标题中的描述信息。
- 详细页本身的 h1 标签。

例子如下所示。

```

<title>中国燃气总经理和执行总裁疑被警方带走_网易新闻中心</title>
<title>干部考核不宜唯“德孝”是举 - 第一视点 - 华声评论 - 华声在线</title>

```

在“-”或“\_”等分隔符前面的文字是真正的标题，但是这样也可能出错，例如某个网页的标题是：“关于《公路工程技术标准》（JTG B01-2003）电子版的说明”。

网页链接中的锚点文本是对目标网页主题内容的概括，所以往往用它作为一个网页的标题描述。图 5-4 显示了来源于新华网的两个页面之间的对应关系。

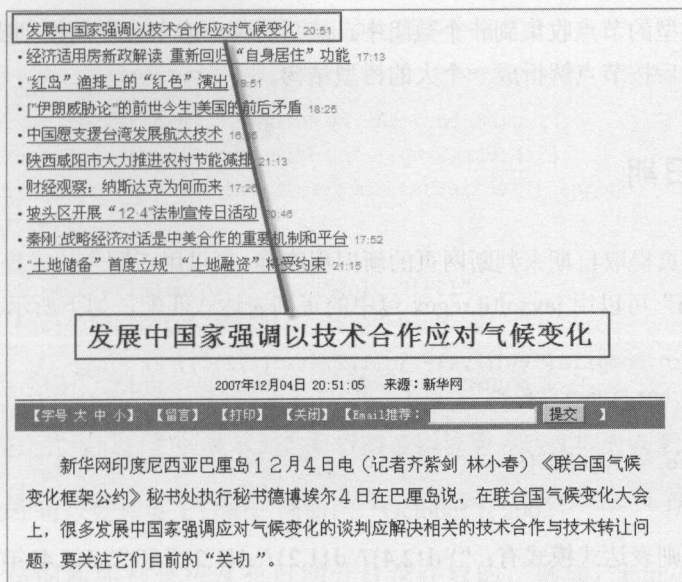


图 5-4 网页标题和锚点文本的对应关系

把列表页上的链接 URL 对应的 标签的“title”属性获取下来，称为列表页描述标题。详细页页面的标题属性称为内部标题。比较列表页描述标题和内部标题，估计出详细页的标题。由于这两个标题的字符串不是全部相同，例如列表页描述标题如果较长就可能以省略号结尾或者末尾截短字，而内部标题可能包含“\_网站名”等冗余信息，所以可以用最长公共子串的方式对两个标题进行相似度判断。

取得了网页标题以后，还可以利用标题信息计算网页的内容和标题之间的距离。

```
public static double getSimiarity(String title,String body){
    int matchNum = 0;
    for(int i=0;i<title.length();++i) {
        if(body.indexOf(title.charAt(i))>=0) {
            ++matchNum;
        }
    }
    double score = (double)matchNum/( (double)title.length() );
    return score;
}
```

可以对每个文本类型的节点根据内容分类，例如，标题、正文、来源，控制页面的文本或者广告文本，版权信息文本等。



把所有文本类型的节点收集到一个数组中，然后对每一个节点应用规则提取内容，并且组织成树型结构，最后按节点解析成一个大的树型结构。

### 5.2.13 提取日期

经常需要从网页提取日期来判断网页的新旧程度等，可以用正则表达式提取网页中的日期。例如，“2009-12-6”可以用 `java.util.regex` 包中的正则表达式匹配，如下所示。

```
Pattern p = Pattern.compile("\\d{2,4}-\\d{1,2}-\\d{1,2}");
Matcher m = p.matcher(inputStr);
if(m.find()){
    String strDate = m.group();
}
```

其他的一些正则表达式模式有：“`\\d{2,4}/\\d{1,2}/\\d{1,2}`”和“`\\d{2,4}年\\d{1,2}月\\d{1,2}日`”，以及“`\\d{2,4}\\d{1,2}\\d{2,4}`”。日期格式很灵活，用一个正则表达式模式不能完全覆盖，可以考虑使用多个有限状态机做运算。

有时候需要从“Thu, 01 Nov 2012 09:19:59 GMT”这样的字符串中得到日期类型。使用 `DateFormat` 的 `parse` 方法实现提取日期。

```
DateFormat formatter = new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss");
Date date = (Date) formatter.parse("2002.01.29.08.36.33");
String s = formatter.format(date);
System.out.println(s);

formatter = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss Z");
date = (Date) formatter.parse("Tue, 29 Jan 2002 22:14:02 -0500");
s = formatter.format(date);
System.out.println(s);
```

如果有多个格式，这样的写法很麻烦。

```
try {
    return DateTimeFormat.forPattern("yyyy/MM/dd HH:mm:ss").parseDateTime(datePart);
} catch (IllegalArgumentException e) {
    return DateTimeFormat.forPattern("E, MMM dd, yyyy HH:mm").parseDateTime(datePart);
}
```

使用 <http://joda-time.sourceforge.net/> 可以同时处理多个格式。

```
DateTimeParser[] parsers = {
    DateTimeFormat.forPattern( "yyyy-MM-dd HH" ).getParser(),
    DateTimeFormat.forPattern( "yyyy-MM-dd" ).getParser() };
DateTimeFormatter formatter = new DateTimeFormatterBuilder().append( null,
parsers ).toFormatter();

DateTime date1 = formatter.parseDateTime( "2010-01-01" );
DateTime date2 = formatter.parseDateTime( "2010-01-01 01" );
```

这个项目的源代码位于 <https://github.com/JodaOrg/joda-time> 上。

用多个正则表达式提取日期效率不高，可以改成用提取表达式同时匹配和提取。

BIU Normalizer 可以处理英文日期，例如，“October 1, 2002” → “1/10/2002”。

很多新闻网站的新闻都是按发布日期分目录存放的，例如，<http://finance.sina.com.cn/g/20101226/09339163619.shtml> 是 2010 年 12 月 26 日的新闻，所以还可以从 URL 地址上提取日期。

可以利用子网页的日期推断父网页的日期，或者利用父网页的日期推断子网页的日期。

## 5.2.14 提取模板

舆情转载分析中需要从新闻网页中提取新闻来源。

最简单的方法，可以用正则表达式提取。比正则表达式更容易使用的方法是提取模板，即通过标注出要提取的文本对应的类型来实现提取。以产生式规则“来源：{source}”为例，词典放入两个基本词：“来源：”和“{source}”。“来源：”编号 1，“{source}”编号 2，“”编号 3。规则树增加规则序列{1,2,3}=>{1,"UNKNOWN",3}。

左边的词类型序列是：{1,2,3}。右边的词类型序列是：{1,"UNKNOWN",3}。把右边的词类型序列组织成 Trie 树。

提取模板可以同时匹配多个提取表达式，使用代码如下所示。

```
String text = "办公地点：石景山区金顶街五区 3 号（金顶街派出所院内）。乘车路线：乘坐 336 路、977 路、运通 112 路、运通 116 路、959 路、325 路公交车金顶南路站下车，向北 40 米。办公时间：星期一至星期六（法定节假日除外）上午 9:00—12:00；下午 14:00—17:30；";
```

```

ArrayList<String> grammarContent = new ArrayList<String>();
grammarContent.add("办公地点[:|:] {办公地点}。");
grammarContent.add("乘车路线[:|:] {乘车路线}。");
grammarContent.add("办公时间[:|:] {办公时间} ");
StructExtractor se = new StructExtractor(grammarContent);

HashSet<String> keys = new HashSet<String>();
keys.add("办公地点");
keys.add("乘车路线");
keys.add("办公时间");

//根据模板匹配文本提取出感兴趣的内容
HashMap<String,String> data = se.match(text,keys);

for(Entry<String, String> e:data.entrySet()){
    System.out.println(e);
}

```

从网页源代码中直接提取信息的例子。

```

String url = "http://www.bjepb.gov.cn/bjepb/323496/330642/331424/index.html";
String content = HttpUtil.getHtml(new URL(url));

ArrayList<String> grammarContent = new ArrayList<String>();
grammarContent.add("许可条件: </strong>{许可条件} [<strong>|</div>]");
grammarContent.add("收费标准及依据: </strong>{收费标准及依据} [<strong>|</div>]");
grammarContent.add("许可事项名称: </strong>{许可事项名称} [<strong>|</div>]");
StructExtractor se = new StructExtractor(grammarContent);

HashSet<String> keys = new HashSet<String>();
keys.add("许可条件");
keys.add("收费标准及依据");
keys.add("许可事项名称");

//根据模板匹配文本提取出感兴趣的内容
HashMap<String,String> data = se.match(content,keys);

for(Entry<String, String> e:data.entrySet()){
    Document doc = Jsoup.parse(e.getValue());
}

```



```
String text = doc.text();
System.out.println(e.getKey()+">"+text); //输出提取出的键/值对
}
```

### 5.2.15 提取 RDF 信息

在图书馆里，每一本书都要被编目，这样方便查找和使用，网上所有的资源也需要编目。

如果要对网络资源编目，首先就必须有一套编目规则。资源描述框架（Resource Description Framework, RDF），就是一套 W3C 提出的描述网络资源的方法。

遍历 DOM 树，在注释中寻找 RDF，在锚点中寻找版权信息。

### 5.2.16 网页解析器原理

FSA 由状态（state）和转移（transition）两部分构成。根据状态转移的可能性，状态机又分为 DFA（确定有限状态机）和 NFA（非确定有限状态自动机）。

开始 a 标签，文字内容，结束标签。

开始 P 标签，文字内容，结束标签。

用图形象地表示有限状态机，每个状态用一个圆圈表示，状态之间的转换用一条边表示，边上的说明文字是输入事件，形成的图如图 5-5 所示。其中双圈节点表示可以作为结束节点，箭头指向的节点是开始节点。开始节点只能有一个，而结束节点可以有多个。

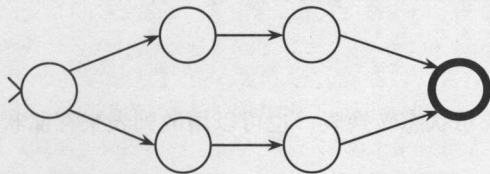


图 5-5 HTML 解析中的有限状态机

转换函数，一般记作  $\delta$ 。转换函数的参数是一个状态和输入符号，返回一个状态。一个转换函数可以写成  $\delta(q, a) = p$ ，这里  $q$  和  $p$  是状态，而  $a$  是一个输入符号。转换函数的含义是：

如果有限状态机在状态  $q$ ，而接收到输入  $a$ ，则有限状态机进入状态  $q$ 。这里的  $q$  可以等于  $p$ 。

例如，图 5-5 中的有限状态机用转换函数表示是： $\delta(\text{Start}, \langle a \rangle) = a$  标签开始； $\delta(\text{Start}, \langle p \rangle) = p$  标签开始； $\delta(\text{文本}, a \text{ 标签结束}) = \text{End}$ ； $\delta(\text{文本}, p \text{ 标签结束}) = \text{End}$ 。

用表 5-1 所示的状态转换表来记录转换函数。状态转换表中的每行表示一个状态，每列表示一个输入字符。

表 5-1 状态转移表

状态/输入	0	1	2
Start		中文	英文
中文	End		
英文	End		
End			

用一个二维数组来记录状态转换表，第一个维度是所有可能的状态，第二个维度是所有可能的事件，二维数组中的值就是目标状态。有限状态机定义如下。

```
public class FSM { //有限状态机
    static State[][] transTable= new State[State.values.length][10] //状态转换表

    static{ //初始化状态转换表
        transTable[State.start.ordinal()][1] = State.chinese; //普通话请按 1
        transTable[State.start.ordinal()][2] = State.english; // press two for english
        transTable[State.chinese.ordinal()][0] = State.end;
        transTable[State.english.ordinal()][0] = State.end;
    }
    State current = State.start; //开始状态
    State step(State s,char c){ //转换函数
        return transTable[s.ordinal()][c - '0'];
    }
}
```

除了使用二维数组来表示状态转换表，也可以用散列表来存储状态转换表。

测试这个有限状态机。

```
FSM fsm = new FSM();
System.out.println(fsm.step(fsm.current, '1')); //输出 chinese
```

如果存在从同一个状态接收同样的输入后可以任意到达多个不同的状态，这样的有限状态机叫作非确定有限状态机。从一个状态接收一个输入后只能到达某一个状态，这样的有限状态机叫作确定有限状态机。

FST 根据输入字符串得到另外一个字符串。这里是根据输入字符串得到一个树状结构，也就是 DOM 树。这里的输入字符串是 HTML 源代码。

Jsoup 里使用了状态模式来实现状态机。状态模式是设计模式的一种，它将状态和对应的行为绑定在一起。而在状态机的实现过程中，使用它来实现状态转移时的处理再合适不过了。

### 5.3 RSS

因为 XML 比 HTML 更规范，所以出现了 XML 格式封装的数据源。RSS (Really Simple Syndication) 是对网站栏目的一种 XML 格式的封装。一些论坛、博客或者新闻网站提供 RSS 格式的输出，方便程序快速访问更新的信息。RSS 抓取的第一步是解析 RSS 数据源。

为了处理非 UTF-8 编码的 RSS 源，首先用 HttpClient 下载 XML 内容。头部 Content-Type 中包含了编码信息。RSS 内容样例如下。

[illegible]



```
</item>
```

定义对应的 POJO 类。

```
public class RssItem {
    public URL url;
    public URL comments;
    public String title;
    public String desc;
    public String author;
    public Date postdate;
    public Date replydate;

    public int replyNum=0;
    public int readNum=0;
}
```

### 5.3.1 Jsoup 解析 RSS

虽然 Jsoup 不是设计成用来解析 RSS 包的，但实际上可以使用它解析 RSS。

Jsoup 代码如下所示。

```
final String url = "http://www.newsmth.net/nForum/rss/board-ITExpress"; //RSS 源
String content = HttpUtil.getHtml(new URL(url));
Document doc = Jsoup.parse(content);
for( Element item : doc.select("item") ) { //选择出所有的项目
    final String title = item.select("title").first().text(); //选择出'title'
    final String link = item.select("link").first().nextSibling().toString().trim(); //选
    择'link'
    String desc =
        StringEscapeUtils.unescapeHtml4(item.select("description").first().toString());
    final Document descr =Jsoup.parse(desc);

    System.out.println(title);
    System.out.println(link);
    System.out.println(descr);
    System.out.println();
}
```

循环里 select 耗费 CPU 和内存？item.select("title") 限定范围找更快。

## 5.3.2 ROME

ROME (<https://github.com/rometools/rome>) 是一个常用的 RSS 解析包。

首先增加 `rome-1.0.jar` 到项目中。因为 ROME 依赖 `jdom` 解析 XML，所以还需要增加 `jdom-1.0.jar` 到项目。

ROME 把 RSS 和 Atom 的联合源表示成 `com.sun.syndication.synd.SyndFeed` 接口的实例。

```
URL feedUrl = new URL("http://www.newsmth.net/nForum/rss/board-ITExpress");
SyndFeedInput input = new SyndFeedInput();
SyndFeed feed = input.build(new XmlReader(feedUrl));
```

也可以使用 `HttpClient` 下载 XML 内容后再把要解析的 XML 传给 `SyndFeed` 对象。

```
String url = "http://www.newsmth.net/nForum/rss/board-ITExpress";
String content = HttpClient.getHtml(new URL(url));

InputStream is = getStream(content);
InputSource source = new InputSource(is);
SyndFeedInput input = new SyndFeedInput();
SyndFeed feed = input.build(source);
```

## 5.3.3 抓取流程

RSS 抓取流程是：首先从网站中自动发现 RSS，然后遍历每一个 RSS，必要的时候分析详细页面。

RSS 种子 (Feed) 就是一个发布最新信息的 URL 地址。为了读取一个 RSS 种子，要先定义读取种子的源，然后定义构建新闻频道对象模型的 `ChannelBuilder`。

```
ChannelBuilder builder = new ChannelBuilder();
String url = "http://rss.news.yahoo.com/rss/topstories"; //RSS 种子
Channel channel = (Channel) FeedParser.parse(builder, url);
System.out.println("标题: " + channel.getTitle());
System.out.println("描述: " + channel.getDescription());
System.out.println("内容:");
for (Object x : channel.getItems()) { //遍历最新的新闻条目
    Item anItem = (Item) x;
    System.out.print("标题:" + anItem.getTitle() + " 描述: ");
```

```

System.out.println(anItem.getDescription());
}

```

如何从网页发现 RSS 种子？下面是一个对 RSS 种子的描述。

```

<link href="http://blog.csdn.net/myth1979/rss.aspx" title="RSS"
type="application/rss+xml" rel="alternate" />

```

根据 `type="application/rss+xml"` 可以确定 `link` 标签中包含的 URI `http://blog.csdn.net/myth1979/rss.aspx` 是 RSS 种子。利用 `HTMLParser` 解析出有效的 Feed 的程序如下所示。

```

TagNode tagNode = (TagNode)node;
String name = ((TagNode)node).getTagName();
if (name.equals("LINK") && !tagNode.isEndTag()) {
    String href=tagNode.getAttribute("HREF");
    if(href!=null &&
        (href.indexOf("rss")>=0 ||
         href.indexOf("feed")>=0 ||
         href.indexOf("rdf")>=0||
         href.indexOf("xml")>=0||
         href.indexOf("atom")>=0)) {
        //验证 Feed 的有效性
        boolean ret = rParser.valid(href);
        if(ret)
            rrsUrls.add(href);
        if("alternate".equals(tagNode.getAttribute("REL"))) {
            outURLs.clear();
            System.out.println("end find");
            return rrsUrls;
        }
    }
}
if( name.equals("A") ) {
    String href = tagNode.getAttribute("HREF");
    if(href != null &&
        (href.indexOf("rss")>=0 ||
         href.indexOf("feed")>=0 ||
         href.indexOf("rdf")>=0||
         href.indexOf("xml")>=0||
         href.indexOf("atom")>=0)) {
        boolean ret = rParser.valid(href);
    }
}

```



```

        if(ret)
            rrsUrls.add(href);
    }
    if(href != null &&
        outURLs!=null &&
        href.startsWith("http://") &&
        href.indexOf(domain)>=0) {
        outURLs.add(href);
    }
}

```

从一个网页发现 RSS 种子的步骤如下所示。

- (1) 首先，用一个方法验证种子是否有效。
- (2) 如果这个 URI 已经指向一个种子，则只是返回它，否则分析这个页面。
- (3) 看这个页面的头信息是否包含 link 标签。
- (4) <A>链接到同一个服务器上以 “.rss” “.rdf” “.xml” 或者 “.atom” 结尾的种子。
- (5) <A>链接到同一个服务器上包含 “.rss” “.rdf” “.xml” 或者 “.atom” 的种子。
- (6) <A>链接到以 “.rss” “.rdf” “.xml” 或者 “.atom” 结尾的外部服务器种子。
- (7) <A>链接到包含 “.rss” “.rdf” “.xml” 或者 “.atom” 的外部服务器种子。
- (8) 尝试猜测一些可能有 Feed 的通用的地方，例如 index.xml、atom.xml 等。

## 5.4 网页去噪

一个页面中经常包括导航栏或底部的公司介绍等信息，这样的信息在很多页面都会出现，可以看作噪声信息。此外还有些广告信息。

详细页的网页去噪和目录页的网页去噪实现方法不同，只需要从目录页提取详细页面就可以了。本节主要讨论详细页的网页去噪问题。

把锚点中的文字看成是无效文字。

```
<a href="http://news.zol.com.cn/more/2_1485.shtml">行业新闻</a>
```

导航栏的特征如下所示。

- 锚文本是短文本，没有标点符号，往往链接到其他子域名或者其他子目录。非锚文本很少。
- 可能包括小的 IMG 标签。

推荐新闻栏目特点是，锚文本是长文本，一般没有标点符号。非锚文本很少。

一般情况下可以去掉网页 DOM 树中的 FORM、Select、IFRAME、INPUT、STYLE 中的节点。

一般从详细页面去掉网页噪声，噪声特征如下所示。

- 多以链接的形式出现，链接到别的相关页面。
- 有很多锚文本，但标点符号较少，锚文本往往是对其他链接页面的说明。
- 有许多常见的噪声文本，如版权声明等。在视觉上，多出现于网页的边缘。

正文块中往往换行符较少，而 p 节点较多。

网页去噪的基本方法是利用各种通用的特征来区分有效的正文和页眉、页脚、广告等其他信息。其中一个常用的特征是链接文字比率，可以把 HTML 转换成 DOM 树，对每个节点计算链接文字比率。如果该节点的链接文字比率高于 1/4，就把这个节点去掉。

另外，可以根据链接节点中包含的文本长度对节点分类，看长度大于 5 个字的链接节点和文字长度的比例，长度小于 5 个字的链接节点和文字长度的比例。

写 CSS 解析器来判断某个 div 的宽度。写 CSS 解析器比写 JS 解析器容易多了。

## 5.4.1 NekoHTML

首先，用 NekoHTML 递归遍历节点下子树的方法，计算一个节点下的链接数。

```
/**
 * 计算一个节点下的链接数
 *
 * @param iNode 开始计算的节点
 * @return 链接数
 */
```

```

public static int getNumLinks(final Node iNode) {
    int links = 0;
    if (iNode.hasChildNodes()) {
        Node next = iNode.getFirstChild();
        while (next != null) {
            Node current = next;
            next = current.getNextSibling();
            //递归调用计算链接数的方法
            links += getNumLinks(current);
        }
    }

    if (isLink(iNode))
        links++;

    return links;
}

```

计算一个节点下的有效正文长度，忽略锚点上的字。

```

/**
 * 计算一个节点下的单词数
 *
 * @param iNode 开始计算的节点
 * @return 单词数
 */
private int getNumWords(final Node iNode) {
    int words = 0;

    if (iNode.hasChildNodes()) {
        Node next = iNode.getFirstChild();

        while (next != null) {
            Node current = next;
            next = current.getNextSibling();

            //如果当前节点是一个链接，则不往下深入
            if (!isLink(current))
                words += getNumWords(current);
        }
    }
}

```



```

//检查节点是文本节点还是元素节点
int type = iNode.getNodeType();

//文本节点
if (type == Node.TEXT_NODE) {
    String content = iNode.getNodeValue();
    words += getHTMLeLen(content);
}

return words;
}

```

计算一个文本中包括的正文实际长度的方法如下所示。

```

private int getHTMLeLen(String text) {
    int len = 0;
    for(int i=0;i<text.length();++i) {
        if(text.charAt(i)==' ') {

        }
        else if(text.charAt(i)==' ') {

        }
        else if(text.charAt(i)==' ') {

        }
        else if(text.charAt(i)=='&') {
            i+=5;
        } else {
            ++len;
        }
    }
    if( len<10)
        len = 0;
    return len;
}

```

根据链接文字比删除无效节点的过程如下所示。

(1) 计算节点下的链接数。

(2) 计算节点下的文字数。

(3) 计算节点的链接文字比 = 节点下的链接数/节点下的文字数。

(4) 如果节点的链接文字比大于某一个阈值，则删除这个节点。

删除无效节点的实现代码如下所示。

```
/**
 * 如果链接比率合适则删除这个表单元
 * @param iNode 表单元节点
 */
public void testRemoveCell(final Node iNode) {
    //如果这个表单元没有孩子节点，则不处理这个表单元
    if (!iNode.hasChildNodes())
        return;

    double links;//iNode 节点下的链接数
    double words;//iNode 节点下的单词数

    //计算链接和单词数
    links = getNumLinks(iNode);
    words = getNumWords(iNode);

    //计算链接文字比并检查是否被 0 除
    double ratio = 0;
    if (words == 0)
        ratio = settings.linkTextRatio + 1;
    else
        ratio = links / words;

    if (ratio > settings.linkTextRatio) { //如果链接文字比大于指定值，则删除该节点
        iNode.getParentNode().removeChild(iNode);
    }
}
```

假设网页所有的正文都位于一个正文节点中。首先，利用 NekoHTML 生成网页 DOM 树。其次，过滤网页上的所有节点得到有用的节点放入到 `nodeInfList` 数组中。再次，合并 `nodeInfList` 数组中连续的正文节点，使之成为 `nodeInfList` 数组中内容比率最大的节点。最后，通过比较有用节点的内容比率（即单词比率）、节点比率等，选出内容比率最大的节点作为正文节点并返回。正文提取流程如图 5-6 所示。

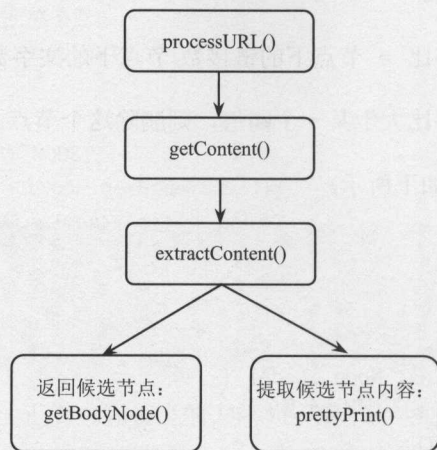


图 5-6 正文提取流程

正文提取主要通过 `ContentExtractor` 类完成，该类的主要方法详细注解如下所示。过滤节点三个函数分别是 `passThroughFilters`、`filterNode` 和 `filterChildren`。

`passThroughFilter` 方法主要用于过滤掉不必要的信息，例如图片链接、广告链接等。通过 `removeChild` 方法移除不必要的孩子节点，`removeEmptyTable` 方法移除空表格节点，`removeAttribute` 方法移除节点的属性。流程是：先判断节点是否为空，不为空就获取节点的名字；再判断特定节点是否有属性，有属性就将属性移除，判断节点是否为文字链接，是否为图片链接节点。

## 5.4.2 Jsoup

首先过滤掉一些没必要的代码，直接用正则替换。过滤了一些常规的无用信息后，再遍历每个子节点，删除噪音节点。这样处理后基本就只有正文内容了，然后抽取其 `text()` 就可以了。

计算一个节点下的链接数。

```
//计算链接数
public static int calcLinks(Element node) {
    Elements links = node.select("a[href]");
    return links.size();
}
```



计算一个节点所包含的文本长度。

```
//计算内容长度
public static double calcWords(Element node) {
    String con = node.text();
    if (con.length() == 0) {
        return 1 + linkTextRadio;
    } else {
        return con.length();
    }
}
```

计算一个节点所包含的标点符号的数量。

```
public static int calcSign(Element node) {
    AVLTree sign = new AVLTree(' ', ',', ';', '.', '\'', '"', '?', '!', ':', ' ', ')'); //把标点符号放入搜索树
    int sum = 0; HttpUtil.getContent
    String text = node.text();
    for(int i=0;i<text.length();++i){
        if(sign.find(text.charAt(i)))
            sum++;
    }
    return sum;
}
```

删除噪音节点的过程：如果该子节点是 a 标签就不进入，直接往下走。如果是一个没有文本信息的节点，则认为该节点是无用的，直接删除这个节点，然后判断是不是节点里有标点符号，一般的页面信息只在正文里有标点符号，对于没有标点符号的也直接删除。计算该节点的链接文字比，大于指定的值时也删除，都不是这样情况的直接往下遍历子节点。

```
private static double linkTextRadio = 0.25; //链接文字比
```

//将所有的空节点全部删除

```
public static Element drawCon(Element node) {
    if (node.tagName().equals("a")) {
        //这个就不用进去深入了
        return node;
    }
    int links; //链接数
```

```

double words; //文字长度
double cellRatio;
int signs; //符号出现的情况

Elements nodes = node.children();
for (Element cnode : nodes) {
    if (!cnode.hasText()) {
        //删除这个节点
        cnode.remove();
    } else {
        links = calcLinks(cnode);
        words = calcWords(cnode);
        cellRatio = links / words; //得到链接文字比
        signs = calcSign(cnode); //计算标点符号的数量
        if (signs < 1) {
            //删除没有标点符号的节点
            cnode.remove();
        } else if (cellRatio > linkTextRatio) {
            cnode.remove();
        } else {
            drawCon(cnode); //递归调用
        }
    }
}
return node;
}

```

测试 HtmlExtract 类的实现如下所示。

```

String content = httpClientGet(url); //得到网页内容
Document doc = HtmlExtract.extract(content); //得到标题和正文

```

### 5.4.3 提取正文

根据 class 的名称选取正文块，例如，“<div class="article\_detail">”。选取正文块的代码如下所示。

```

String className="article_detail";
Elements es = doc.getElementsByClass(className);

```

```
String content = es.first().text();
```

根据机器学习理论框架,需要自动发现内容块对应的 class 名称。首先提取网页中所有类名,然后根据名称筛选,最后判断内容块 class 名称是否有效。

提取网页中所有类名的文法。

```
ExtractGrammar g = new ExtractGrammar();
String right = " class=\"<n>{classname}\"";
g.add(right);
```

根据标题定位正文块。

```
Element titleE = TitleFinder.findElement(doc); //找到标题对应的元素
Element articleBlock = titleE.parent(); //找父元素

//收集块中所有的 classname
TextExtractor ie = getExtractor();
AdjList adjList = ie.getLattice(articleBlock.html());
ArrayList<String> classes = adjList.args.get("classname");
```

有些正文在标题下方的 P 标签中。

调用 commons-lang3-3.1.jar 中的 StringUtils.strip 方法去掉全角空格。

## 5.5 从非 HTML 文件中提取文本

在很多网站中,为了查询大量积累下来的文档,需要从 PDF、Word、Rtf、Excel 和 PowerPoint 等格式的文档提取可以描述文档的文字,其中 Word、Rtf、Excel 和 PowerPoint 格式都来自微软。POI 项目 (<http://poi.apache.org/>) 是一个专门处理微软文档格式的开源项目,是一个纯 Java 的实现,可以在 Linux 平台运行。S1000D 是一种 XML 文档格式,主要用于航空航天和国防工业(军品和民品)中的技术出版物。

为了方便搜索文档,可以设计一个通用接口来处理待索引的文档。

```
//处理文档
```



```
public interface IFilter {
    String getTitle(File file);//返回标题
    String getBody(File file);//返回内容
    IDocument getDocument(File file);//返回全部索引信息
}
```

### 5.5.1 PDF 文件

PDF 是 Adobe 公司开发的电子文件格式。这种文件格式与操作系统的平台无关，可以在多数操作系统上通用。

我们经常会遇到这种情况，就是想把 PDF 文件中的文字复制下来，却发现不能复制，因为 PDF 文件的内容可能加密了，那么把 PDF 文件中的内容提取出来就是这一节要解决的问题。现在已经有很多工具可以帮助完成这项任务了，PDFBox (<http://pdfbox.apache.org/>) 就是专门用来解析 PDF 文件的 Java 项目。

下载文件 `pdfbox-app-1.5.0.jar`，然后在 Eclipse 项目中引入这个文件。提取文本只需要如下所示代码。

```
//加载 fileName 代表的 PDF 文件
PDDocument doc = PDDocument.load(fileName);
//用 PDFTextStripper 提取文本
PDFTextStripper stripper = new PDFTextStripper();
//返回提取的文本字符串
String content = stripper.getText(doc);
```

设定提取的页码范围。

```
PDDocument doc=PDDocument.load(fileName); // document
int i=1; //页号，从第一页开始

PDFTextStripper reader = new PDFTextStripper();
reader.setStartPage(i);
reader.setEndPage(i);
String pageText = reader.getText(doc);
```

需要说明的是这个版本支持中文。使用早期版本的 PDFBox 抽取中文 PDF 文本时，可能会

遇到 Unknown encoding for 'GBK-EUC-H'错误。因为 PdfReader 内部默认支持一些中文字体，但 PDFBox 内部却没有对 GBK-EUC-H 字体的支持。

CMap 文件中定义了 CID 数字和字符编码之间的对应关系。PDFReader 会查找 cmap 路径下的 CMap 文件中的 GBK-EUC-H，但是在 PDFBox 中却没法查找。可以在 PDFReader 的文档属性中看到 PDF 文件编码用到的字体。

在 PDFBox 0.8 版本中，会查找操作系统本身的字体。

```
java.awt.Font[] allFonts =
    java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
int numberOfFonts = allFonts.length;
for (int i=0;i<numberOfFonts;i++){
    java.awt.Font font = allFonts[i];
    System.out.println(font);
}
```

GBK-EUC-H 是重新排序后生成的字体，不是操作系统直接支持的全真（true type）字体，所以这种方法仍然找不 GBK-EUC-H 字体。GBK-EUC-H 字体在 Adobe 公司的 CMap 字体文件中定义。因为涉及 Adobe 公司的专利问题，PDFBox 0.8 以后的版本不再包括 Adobe 公司的 CMap 字体文件。使用编译工具 Ant 编译该项目会自动从 Adobe 公司下载 CMap 字体文件，这样就避免了因为直接包含字体文件所带来的法律问题。

PDF 只是一组指令告诉你某个字符应该如何打印或者显示出来，所以从 PDF 提取格式化的文本并不是一件简单的工作。虽然可以抽取 PDF 文件中的文本，但是对于 PDF 文件中的图片 PDFBox 是无能为力的。对于包含文字的图片，需要借助 OCR 软件从图片中识别出文字。

从 PDF 文件提取出来的文本有时候会在数字前面出现多余的空格。例如，<http://www.cninfo.com.cn/finalpage/2015-12-12/1201830823.PDF> 中的“自2015年12月14日开市起停牌”。判断两个字符之间是否有空格，需要调整参数。PDFTextStripper.setSpacingTolerance()方法用来控制是否插入空格，示例如下所示。

```
File pdfFile = new File("f:/stock/蒙草抗旱：关于筹划发行股份购买资产的停牌公告.pdf");
PDDocument doc = PDDocument.load(pdfFile);
PDFTextStripper stripper = new PDFTextStripper();
stripper.setSpacingTolerance(800.0f); //0.08f
```

```
String content = stripper.getText(doc);
System.out.println(content);
```

如果需要修改 PDFTextStripper 的源代码，可以使用 Maven 编译 PDFBox 项目源代码。在编译过程中，有些 jar 包无法下载。Maven 的安装目录下的 conf 文件下有个 settings.xml 文件，编辑该文件，使用镜像网址就可以了。

连不上 Maven 库，可以使用 oschina 提供的镜像，<http://maven.oschina.net/content/groups/public/>。

在 setting.xml 中配置地址如下所示。

```
<mirrors>
  <mirror>
    <id>CN</id>
    <name>OSChina Central</name>
    <url>http://maven.oschina.net/content/groups/public/</url>
    <mirrorOf>central</mirrorOf>
  </mirror>
</mirrors>
```

编译 jar 包忽略测试阶段。

```
mvn install -Dmaven.test.skip=true
```

因为 PDF 文件中有可能存在重叠位置的文字，PDFTextStripper 有对于重叠位置文字的判断。如果两个 TextPosition 位于差不多同样的位置，则不重复抽取重叠的 TextPosition 代表的文字。

```
private boolean overlap( TextPosition tp1, TextPosition tp2 ){
    if(!within(tp1.getHeight(),tp2.getHeight(), 1.1f))
        return false;

    float diff = (tp1.getHeight() + tp2.getHeight())*0.02f;
    return within( tp1.getX(), tp2.getX(), diff) && within( tp1.getY(), tp2.getY(), diff) ;
}
```

在提取 PDF 文件中可搜索的文字的问题解决后，下一个问题是提取标题。例如，在 Google 上搜索“filetype:pdf”可以看到 Google 提取的 PDF 文件标题。

如果 PDF 元数据中已经存储了文档标题，可以通过元数据取得文档标题。

```
//取得文档元数据
```



```
PDDocumentInformation info = document.getDocumentInformation();
this.title = info.getTitle();
```

但是很多 PDF 文件中元数据并没有存储任何内容。所以在大多数情况下，仍然需要从内容中分析出标题。

可以利用文本的位置或字体等信息帮助选取标题。例如，在首页字体最大的文字有可能是 PDF 文件的标题。可以通过 `TextPosition` 对象的 `getFontSizeInPt` 方法返回文字字体大小。

为了取得文本的位置信息和颜色等，需要更加深入地了解 `PDFBox` 的运行原理。`PDF` 规范可以从 [http://www.adobe.com/devnet/pdf/pdf\\_reference.html](http://www.adobe.com/devnet/pdf/pdf_reference.html) 上下载。`PDF` 内容流中包含许多操作符（Operator）。在 `PDFBox` 中，每种操作符都有专门对应的处理类，记录在配置文件 `Resources/PageDrawer.properties` 中。

`BT=org.apache.pdfbox.util.operator.BeginText`，表示 `BT` 操作的处理类是 `org.apache.pdfbox.util.operator.BeginText`。`BT` 表示开始一个文本对象，而 `ET` 表示结束一个文本对象。

`PDFBox` 中的应用程序 `PageDrawer` 实现输出 `PDF` 文件到可视化窗口。`PageDrawer` 有完整的对于文字显示方面的处理。为了提取标题，可以通过在 `PageDrawer` 的实现基础上，简化一些与 `Path` 和 `Line` 相关操作符的处理，只把位置信息和颜色等信息提取出来。

文字的颜色信息并不能直接得到，而是依赖于上下文。`PageDrawer` 类包含了对图像的处理。

```
if( this.getGraphicsState().getTextState().getRenderingMode() ==
    PDTextState.RENDERING_MODE_FILL_TEXT ){
    graphics.setColor( this.getGraphicsState().getNonStrokingColorSpace().createColor() );
}
else if( this.getGraphicsState().getTextState().getRenderingMode() ==
    PDTextState.RENDERING_MODE_STROKE_TEXT ) {
    graphics.setColor( this.getGraphicsState().getStrokingColorSpace().createColor() );
}
```

## 5.5.2 Word 文件

Word 是微软公司开发的字处理文件格式，以“doc”或者“docx”作为文件后缀名。Apache 的 POI (<http://poi.apache.org/>) 可以用来在 Windows 或 Linux 平台下提取 Word 文档。用 POI

提取文本的基本方法如下所示。

```
public static String readDoc(InputStream is) throws IOException{
    //创建 WordExtractor
    WordExtractor extractor=new WordExtractor(is);
    //对 doc 文件进行提取
    return extractor.getText();
}
```

为了提取 Word 文档的标题，需要深入了解 POI 接口。一个 Word 文档包含一个或者多个 Section，每个 Section 下面包含一个或者多个 Paragraph，每个 Paragraph 下面包含一个或者多个 CharacterRun，如图 5-7 所示。

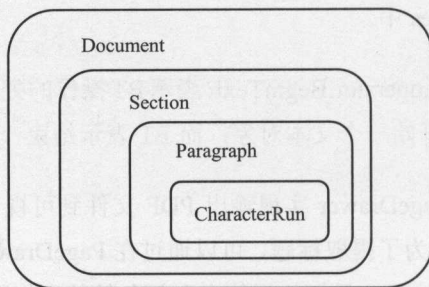


图 5-7 Word 文档结构图

遍历 Word 文档结构的代码如下所示。

```
Range r = doc.getRange();

for (int x = 0; x < r.numSections(); x++) {
    Section s = r.getSection(x);
    for (int y = 0; y < s.numParagraphs(); y++) {
        Paragraph p = s.getParagraph(y);
        for (int z = 0; z < p.numCharacterRuns(); z++) {
            CharacterRun run = p.getCharacterRun(z);
            //字符串文本
            String text = run.text();
            System.out.println(text);
        }
    }
}
```

对于 Word 95 这样的老版本，需要使用 Word6Extractor。

```
Word6Extractor extractor = new Word6Extractor(in);
String text = extractor.getText();
```

### 5.5.3 Rtf 文件

1992 年，为了定义简单的格式化的文本和嵌入的图片，微软公司引入了富文本格式 (Rtf)。最开始是为了在不同的操作系统（例如，MS-DOS、Windows 和 OS/2 以及苹果的 Macintosh）上的不同的应用程序之间转移数据，现在这种格式已经广泛用于 Windows 系统，因为它可以在 RichTextBox 控件中编辑。

Rtf 的版本从 1.0 到 1.9。Rtf 是 8 位格式的，为了方便传输，标准的 Rtf 文件只由 ASCII 字符组成，中文字符用转义符来表示。每个 Rtf 文件都是一个文本文件。文件开始处是 {rtf，它作为 Rtf 文件的标志是必不可少的。Rtf 阅读器根据它来判断一个文件是否为 Rtf 格式，然后是文件头和正文，文件头包括字体表、文件表、颜色表等几个数据结构，正文中的字体、表格的风格就是根据文件头的信息来格式化的。每个表用一对大括号括起来，其中包含很多用字符 “\” 开始的命令。举个最简单的 Rtf 文件的例子，文件中只包含一行文字：{rtf1foobar}。用写字板打开这个文件，显示：“foobar”。

许多开源的 Rtf 文件解析器不能正确处理多字节编码内容，例如，javax.swing.text.rtf。当然有的项目也能够处理包含 Unicode 编码的 Rtf 文件，例如，RtfConverter（下载地址是 <http://www.codeproject.com/KB/recipes/RtfConverter.aspx>），不过这个项目是 C# 实现的，现在介绍如何用 Java 实现一个 Rtf 文件解析器 RtfConverter4J。

Rtf 文件解析器 RtfConverter4J 的设计目标如下。

- 可以在各层次上分析 Rtf 数据。
- 把对 Rtf 数据的解析和解释分开。
- 保持解析器和解释器的可扩展性。
- Rtf 转换应用程序容易上手。
- 采用开放式架构，能够很容易地自定义 Rtf 转换器。

因此，设计了如图 5-8 所示的开放式架构。



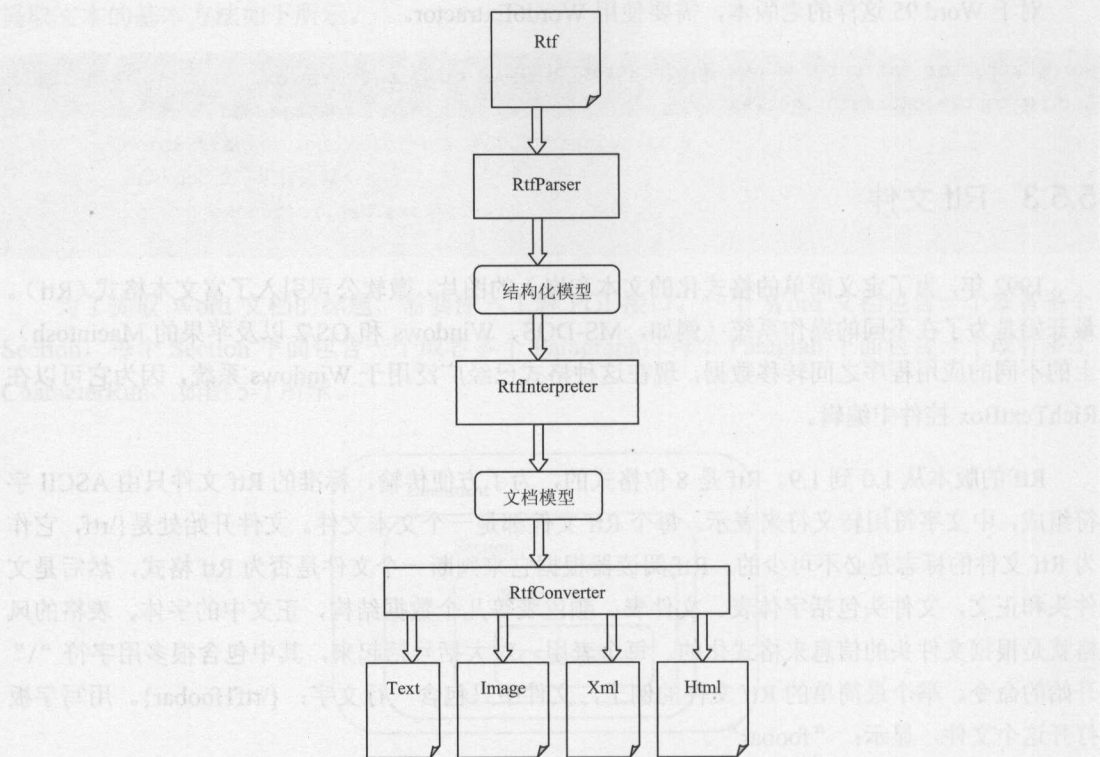


图 5-8 Rtf 转换程序总体结构图

**RtfParser** 类用于完成实际的数据解析。除了识别标签，也处理字符编码，支持 Unicode。**RtfParser** 把 Rtf 数据分成如下几种基本元素。

- **RtfGroup**: Rtf 元素的集合。
- **RtfTag**: 一个 Rtf 标签的名字和值对。
- **RtfText**: 任意的文本内容，但是文本内容不一定是可见的。

在 **RtfParser** 中对 Unicode 编码的处理代码如下所示。

```
if (tagName.equals(RtfSpec.TagUnicodeCode)) {  
    //读入Unicode编码的字符的值  
    int unicodeValue = tag.getValueAsNumber();  
    char unicodeChar = (char) unicodeValue;  
    this.curText.append(unicodeChar);  
    for (int i = 0; i < this.unicodeSkipCount; i++) {
```

```

//跳过指定数量的文本
char skip1 = (char) reader.read();
if (skip1 == ' ') {
    char skip2 = (char) PeekNextChar(reader, false);
    if (skip2 == '\\') {
        reader.read();
        char skip3 = (char) PeekNextChar(reader, false);
        while (skip3 != '\\' && skip3 != '}' && skip3 != '{') {
            reader.read();
            skip3 = (char) PeekNextChar(reader, false);
        }
    }
} else if (skip1 == '\\') {
    reader.read();
    char skip3 = (char) PeekNextChar(reader, false);
    while (skip3 != '\\' && skip3 != '}' && skip3 != '{') {
        reader.read();
        skip3 = (char) PeekNextChar(reader, false);
    }
} else if (skip1 == '\r') {
    reader.read();
    char skip2 = (char) PeekNextChar(reader, false);

    if (skip2 == '\\') {
        reader.read();
        char skip3 = (char) PeekNextChar(reader, false);
        while (skip3 != '\\' && skip3 != '}' && skip3 != '{') {
            reader.read();
            System.Console.Write((char) reader.Read());
            skip3 = (char) PeekNextChar(reader, false);
        }
    }
}

} else if (tagName.Equals(RtfSpec.TagUnicodeSkipCount)) {
    //读入 UnicodeSkipCount 的值
    int newSkipCount = tag.GetValueAsNumber();
    if (newSkipCount < 0 || newSkipCount > 10) {
        throw new Exception("invalid unicode skip count: " + tag);
    }
    this.unicodeSkipCount = newSkipCount;
}

```

因为 Rtf 数据中可能包含另外一种十六进制表示的 Unicode，所以增加了对 TagUnicodeSkipCount 的处理。

Rtf 文件解析器的结构如图 5-9 所示。实际的解析过程可以通过 ParserListener 监听。ParserListener 通过观察者模式提供了对某个事件反应的机会并执行相应的动作。系统已经集成的解析监听器 RtfParserListenerFileLogger 可以用来把 Rtf 元素的结构写入日志文件（主要用于开发阶段的调试）。输出可以通过 RtfParserLoggerSettings 配置。

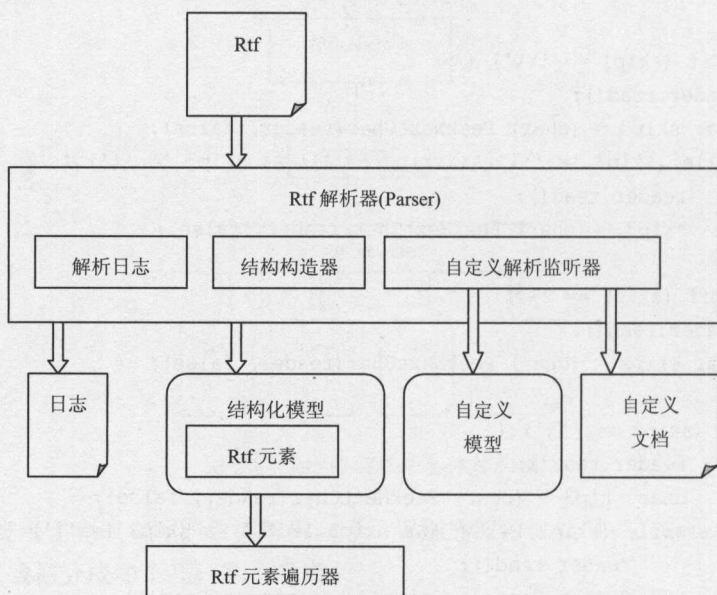


图 5-9 Rtf 文件解析器的结构

RtfParserListenerStructureBuilder 根据解析过程中碰到的 Rtf 元素生成结构化模型。这个模型把基本元素表示成 IrtfGroup、IrtfTag 和 IrtfText 的实例，可以通过 RtfParserListener-StructureBuilder.StructureRoot 取得层次结构。

当 Rtf 文档解析成结构化模型后，就可以通过 Rtf 解释器来解释。Rtf 文件解释器的结构化模型如图 5-10 所示。解释结构化模型的一个方法是构造文档模型来提供对文档内容意义的高层次抽象。一个简单的文档模型由 6 个部分组成：文档信息，其中包括标题、主题和作者等；用户属性；颜色信息；字体信息；文本格式；可视化信息。其中，可视化信息包括以下内容。



- 文本相关的格式化信息。
- 分隔符。线、段落、节、页。
- 特殊字符。制表符、段落开始/结束、下画线、空格、圆点、引号、连字符。
- 图片。

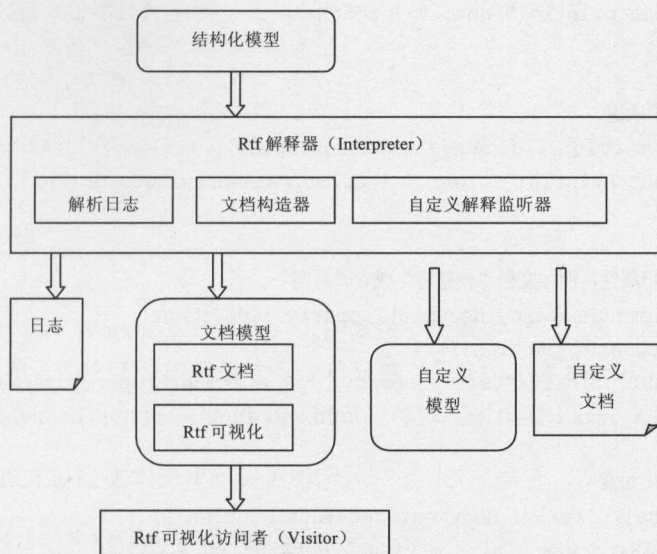


图 5-10 Rtf 文件解释器的结构化模型

下面的例子展示了如何使用文档模型的高层 API。

```

//显示 Rtf 文件内容
public static void RtfWriteDocumentModel(String rtfStream) throws Exception {
    RtfInterpreterListenerFileLogger logger = null;
    IRtfDocument document = RtfInterpreterTool.BuildDoc(rtfStream, logger);
    RtfWriteDocument(document);
} //RtfWriteDocument 模型 1

//根据文档模型遍历
public static void RtfWriteDocument(IRtfDocument document) {
    System.out.println("RTF Version: " + document.getRtfVersion());

    //显示文档信息
    System.out.println("Title: " + document.getDocumentInfo().getTitle());
  
```

```
System.out.println("Subject: "
    + document.getDocumentInfo().getSubject());
System.out.println("Author: " + document.getDocumentInfo().getAuthor());

//显示字体表中的字体
for (String fontName : document.getFontTable().keySet()) {
    System.out.println("Font: " + fontName);
}

//显示颜色表中的颜色
for (IRtfColor color : document.getColorTable()) {
    System.out.println("Color: " + color.getAsDrawingColor());
}

//取得文档的用户属性,例如文档“创建者”或者“时间”
for (IRtfDocumentProperty documentProperty : document
    .getUserProperties()) {
    System.out.println("User property: " + documentProperty.getName());
}

//遍历所有可视化元素
for (IRtfVisual visual : document.getVisualContent()) {
    RtfVisualKind visualKind = visual.getKind();
    if (visualKind == RtfVisualKind.Text) { //文本
        System.out.println("Text: "
            + ((IRtfVisualText) visual).getText());
    } else if (visualKind == RtfVisualKind.Break) { //换行符号
        System.out.println("Tag: "
            + ((IRtfVisualBreak) visual).
                getBreakKind().toString());
    } else if (visualKind == RtfVisualKind.Special) { //特别字符
        System.out.println("Text: "
            + ((IRtfVisualSpecialChar) visual).getCharKind()
                .toString());
    } else if (visualKind == RtfVisualKind.Image) { //图像
        IRtfVisualImage image = (IRtfVisualImage) visual;
        System.out.println("Image: " + image.getFormat().toString()
            + " " + image.getWidth() + "x" + image.getHeight());
    }
}
```

调用 RtfConverter4J 提取文本。

```
URL u = new URL("http://www.silo.net/LaReja-2005-05-07/texto_chino_7M.rtf");
//创建一个URL连接对象
URLConnection uc = u.openConnection();
//提取内容并输出
InputStream is = uc.getInputStream();
RtfExtractor extractor=new RtfExtractor(is);
String text = extractor.getText();
is.close();
System.out.println("text:"+text);
```

### 5.5.4 Excel 文件

Excel 文件由工作簿 (Workbook) 组成。工作簿由一个或多个工作表 (Sheet) 组成, 每个工作表都有自己的名称, 每个工作表又包含多个单元格 (Cell)。除了 POI 项目, 还有开源项目 jxl (<http://www.andykhan.com/jexcelapi/index.html>) 可以用来读写 Excel 文件。

调用 Apache 的 POI 提取文本的代码如下所示。

```
public static String readDoc(InputStream is) throws IOException{
    //读入 Excel 文件数据流
    ExcelExtractor extractor = new ExcelExtractor(new POIFSFileSystem(is));
    //不返回公式
    extractor.setFormulasNotResults(true);
    //不包括 Sheet 名称
    extractor.setIncludeSheetNames(false);
    return extractor.getText();
}
```

用 JDBC 访问 Excel 文件中的结构化信息, 把每个 Sheet 看成一个表。

```
Class.forName("com.googlecode.sqlsheet.Driver");
String path ="D:/search/全文检索/文件级 EXCEL.xls";
Connection conn = DriverManager.getConnection("jdbc:xls:file:"+path);
```

解决支持中文问题。下载 JSqlParser 源代码 <https://github.com/JSQParser/JSqParser>。修改 JSqlParserCC.jj 文件, 扩展 LETTER 可以接收的字符范围。

```
< #LETTER: ["a"-"z", "A"-"Z", "_", "\u0024",
```



```
"\u0041"-" \u005a",
"\u005f",
"\u0061"-" \u007a",
"\u00c0"-" \u00d6",
"\u00d8"-" \u00f6",
"\u00f8"-" \u00ff",
"\u0100"-" \u1fff",
"\u3040"-" \u318f",
"\u3300"-" \u337f",
"\u3400"-" \u3d2d",
"\u4e00"-" \u9fff",
"\uf900"-" \ufaff"] >
```

用 javacc 生成 Java 文件。

```
D:\javacc-6.0\bin>java -cp .\lib\javacc.jar javacc JSqlParserCC.jj
```

把生成出来的文件放到 JSqlParser 源代码中，打包即可。

### 5.5.5 PowerPoint 文件

在 PowerPoint 视图编辑区的上半部分显示幻灯片的缩略图，下半部分是备注编辑区，所以可提取的文本包括幻灯片显示的文本和备注中的文本。调用 Apache 的 POI 提取 PowerPoint 文件中的文本代码如下所示。

```
public static String readDoc(InputStream is) throws IOException{
    //创建 PowerPointExtractor
    PowerPointExtractor extractor=new PowerPointExtractor(is);
    //取得所有的幻灯片文本，但是不包括备注中的文本
    //如果要返回备注中的文本，可以调用 setNotesByDefault(true)
    return extractor.getText();
}
```

## 5.6 提取标题

5.5 节只介绍了如何从各种格式的文档中提取文本。有很多文件名的命名没有意义，例如类

似“20090224153208138.pdf”由数字组成的文件名。在搜索结果中显示一个有意义的文件标题而不是文件名，能够改进用户的搜索体验。提取标题是信息结构化处理的第一步，首先从一般意义上来讨论从文件内容提取标题的方法，然后按文件类型分别实现提取标题。

### 5.6.1 提取标题的一般方法

提取标题的流程如图 5-11 所示。

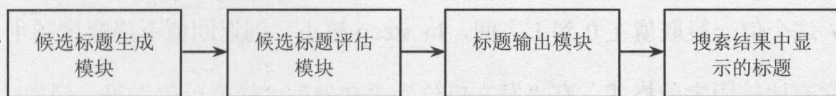


图 5-11 提取标题的流程

- 候选标题生成模块：首先从提取出来的原子文本组织成文档结构树。构建文档结构树既可以用自下而上的方法，从原子节点构造起；也可以用自上而下的方法，首先从根节点将文字划分成大的单元，然后逐步从大块文字细分。当采用自下而上的方法时，如果文字在同一行，并且字体、字体大小、颜色都一致，则视为不可拆分。然后根据字体、字体大小、颜色、位置等信息再次合并文字。从文档结构树给出几个可能的候选标题。
- 候选标题评估模块：对每个候选标题，按照特征打分。可以把要用到的特征都作为候选标题类的属性。其中，最主要的特征是标题字符串，此外还有字符串所在位置、字体大小、字体颜色等信息。也可以根据标题字符串对整个文章的概括程度、通顺性、意义完整性打分。从对文章的概括程度考虑，可以按照  $TF*IDF$  等方法选取重要性较高的词作为关键词，然后根据关键词对每个候选标题字符串给出可能性权重。还可以比较候选标题字符串和首页中的其他文字，看候选标题相对其他文字的代表度或者可替代性，也就是候选标题对其他文字的覆盖度。从通顺性与意义完整性考虑，可以考虑准备一个标题语料库，提取出词法规则和作为标题常用的搭配规则，也可以对大量标题训练一个 HMM 模型，或者用信息提取的方法来评估候选标题字符串。
- 标题输出模块：把权重最大的候选标题挑出来，按照可读的方式输出。

作为标题的文字长度往往既不是太长，也不会太短，大部分标题的长度在 10 到 30 个字之间。超长的标题往往会被搜索引擎截短，网站制作者为了优化搜索排名，也倾向于采用这样的标题长度。可以用 `sweetSpot` 来对候选标题的长度打分。

```
private static int ln_min = 10; //标题最短长度
```

```
private static int ln_max = 30; //标题最长长度
private static float ln_steep = 0.5f;

//如果长度在 10 到 30 之间, 则 sweetSpotScore 返回值是 1
//否则返回值会低于 1, 返回值随长度降低的程度由 ln_steep 决定
public static double sweetSpotScore(int length) {
    return (float) (1.0f /
        Math.sqrt((ln_steep * (float) ( Math.abs(length - ln_min)
            + Math.abs(length - ln_max) - (ln_max - ln_min)) ) + 1.0f) );
}
```

`ln_steep` 这个值一般取值在 0 到 1 之间, `ln_steep` 越小, 则返回值下降幅度越小。

政府公文有比较固定的格式, 有“发文单位”“文号”“接收单位”等。例如, 图 5-12 中的文件“发文单位”是“合肥市物价局文件”, “文号”是“合价服〔2009〕219 号”, “接收单位”是“三县四区物价局”。



图 5-12 政府公文

判断是否发文机关。

```
public static boolean isProSendGovUnit(String s, Color tcolor){
    double pro = 0;

    if(tcolor.equals(Color.RED)) {
        pro+=0.2;
    }

    if(s.endsWith("文件")){
```



```

        pro+=0.4;
    }

    int numBlank = 0;
    int numGovUnit = 0;
    for(int i=0; i<s.length(); i++) {
        char c = s.charAt(i);
        if(c == ' '){
            numBlank ++;
        }
        else if(c == '厅' ||
                c == '所' ||
                c == '局' ||
                c == '会' ||
                c == '委' ||
                c == '府')
            numGovUnit++;
    }
    if(numBlank > 0) {
        pro +=
(0.6*((double)numGovUnit+(double)numBlank)/((double)numBlank+s.length()));
    }
    if(pro > 0.6)
        return true;
    return false;
}

```

判断是否收文机关。

```

public static boolean isProReceiGovUnit(String s,Color tcolor){
    if(!tcolor.equals(Color.BLACK)) {
        return false;
    }

    float pro =0 ;
    if(s.endsWith(": ") || s.endsWith(":")) {
        pro += 0.5;
    }

    for(int i=0; i<s.length(); i++) {
        char c = s.charAt(i);
        if(c == ',' || c == '(' || c == ')' || c == ',') {

```

```

        pro +=0.2;
    }
    else if(c == '厅' ||
           c == '部' ||
           c == '所' ||
           c == '局' ||
           c == '会' ||
           c == '委' ||
           c == '县' ||
           c == '区' ||
           c == '市' ||
           c == '省')
        pro +=0.11;
    }
    pro = pro/s.length();
    if(pro >= 0.07)
        return true;
    return false;
}

```

判断文号，例如“川国税发（2007）065 号”。

```

public static boolean isFileNum(String s,Color tcolor){
    if(!tcolor.equals(Color.BLACK)) {
        return false;
    }
    boolean isSymbol = false;

    //如果碰到开始符号，则匹配对应的结束符号
    for(int index=0; index<s.length(); index++){
        if('[' == s.charAt(index)){
            isSymbol = matchSym(s,index,']');
        }
        else if('(' == s.charAt(index)) {
            isSymbol = matchSym(s,index,')');
        }
        else if ('(' == s.charAt(index)) {
            isSymbol = matchSym(s,index,')');
        }
    }

    return isSymbol;
}

```

```

}

//如果匹配上指定字符则成功退出
public static boolean matchSym(String s,int index,char endMark){
    boolean isSymbol = false;

    boolean markBegSym = true;
    boolean markEndSym = false;
    for(int i =index+1; i<s.length(); i++) {
        if(!markEndSym) {
            if((s.charAt(i)>='0'&& s.charAt(i)<='9')||s.charAt(i) == ' '
                ||(s.charAt(i)>='0'&& s.charAt(i)<='9'))
            {}
            else if(s.charAt(i) == endMark) {
                markEndSym = true;
                i++;
            }
            else {
                markBegSym = false;
                markEndSym = false;
            }
        }
        if(markBegSym && markEndSym) {
            if((s.charAt(i)>='0'&& s.charAt(i)<='9')||s.charAt(i) == ' '
                ||(s.charAt(i)>='0'&& s.charAt(i)<='9'))
            {}
            else if(s.charAt(i) == '号') {
                isSymbol = true;
                index = s.length();
            }
        }
    }

    return isSymbol;
}

```

## 5.6.2 从 PDF 文件中提取标题

用 PdfTitle 表示候选标题。



```

public class PdfTitle {
    public String title = null; //标题文字
    public Color textColor = null; //文字颜色
    public float fontSize; //字体大小
    public float y; //高度
    ArrayList<TextPosition> texts; //包含的文本

    public PdfTitle(String t,float h) {
        title = t;
        y=h;
    }

    public PdfTitle(Color c,float f,float h,ArrayList<TextPosition> text) {
        title = delOverlap(text);
        textColor = c;
        fontSize = f;
        y = h;
        texts = text;
    }
}

```

候选标题单独组成一个段落，或者位于一个段落内部。通过 PDFDegger 可以看到 PDF 文件的树形结构组织。因为通过 PDFStreamEngine 类导出的文字块没有明显的段落信息，所以需要编写程序寻找段落边界。大的字体差别、颜色差别以及文本的垂直位置都可以用来确定是一个独立段落的开始，但有些相对模糊的边界需要更细致的判断。因此把 PdfTitle 设计成可以再次拆分的单元。

(1) 根据文本的垂直位置寻找该段落的最大垂直区隔。

(2) 根据最大垂直区隔拆分 PdfTitle。

(3) 如果没有找到合适的标题，对于拆分出来的新的 PdfTitle 重新应用 (1)、(2) 步骤直到不可再拆分或找到合适的标题为止。

可以通过扩展 org.pdfbox.util.PDFTextStripper 类，覆盖 processTextPosition 方法来遍历 PDF 文件中的 TextPosition 对象。提取最大字体的文字作为标题代码如下所示。

```

float currentFontSize = text.getFontSizeInPt();
if(currentFontSize < biggestFontSize){ //字体不一致，则不是标题文字
    consistent = false;
}

```

```

} else if (currentFontSize > biggestFontSize){ //字体最大，是标题文字
    titleGuess = text.getCharacter();
    biggestFontSize = currentFontSize;
    consistent = true;
} else if (currentFontSize == biggestFontSize
    && consistent
    && !"".equals(text.getCharacter().trim())){
    //字体和以前文字的最大字体一样大，是标题文字
    titleGuess += text.getCharacter();
}

```

PDF 文件分成两类：一类首页没有正文，文字比较少且比较多的部分可能是标题。还有一类，首页有正文，正文上面的可能是标题，也可能是段落标题。首页没有正文的情况，不计算标题对于首页正文的概括性。

政府公文中往往包含一些主题词，可以利用主题词或者文章内容来判断标题，也可以建立一个标题的语料库，例如很多标题都是采用“关于……的通知”这样的形式，可以利用模版匹配的方式来保证标题的完整性。

### 5.6.3 从 Word 文件中提取标题

标题往往是居中对齐的，可以通过 Paragraph 的 `getJustification` 方法得到段落的对齐方式。`getJustification` 的返回值为 0 表示左对齐，1 表示居中对齐，2 表示右对齐，3 表示两端对齐。

可以通过 `CharacterRun` 对象取得文字内容、字体大小、文字颜色等信息。

```

run.text(); //文字内容
run.getFontSize(); //字体大小
run.getColor(); //文字颜色

```

### 5.6.4 从 Rtf 文件中提取标题

提取 Rtf 文件的标题设计流程如下所示。

- (1) 生成候选标题：把字体最大的文字块或者居中对齐的文字块作为候选标题。
- (2) 评估候选标题：根据候选标题所在的位置和候选标题与正文的相似度来评分。
- (3) 输出标题模块：选择分值最大的标题输出。

首先定义候选标题类。

```
public static class TitleInfo {
    public String fontName;           //字体名
    public int fontSize;              //字体大小
    public int position;              //位置信息
    public int mergeTo;               //合并块编号
    public boolean isBold;            //是否粗体
    public String text;               //内容
    public HashMap<String, Double> words; //内容切分结果
    public double weight;             //权重
}
```

提取标题整体流程。

```
public static String getRtfTitle(String fileName) throws Exception {
    StringBuffer content = new StringBuffer(); //用来存储全文内容
    //取得候选标题
    ArrayList<TitleInfo> candidates=getCandidates(fileName,content);

    if (candidates == null || candidates.size() == 0) //没有候选标题
        return "";
    else if (candidates.size() == 1)
        return candidates.get(0).text;
    else {
        //候选标题评分
        rankTitle(candidates, content.toString());
        //把评分最高的候选标题作为标题提取结果
        return getBestTitle(candidates);
    }
}
```

取得候选标题部分实现代码。

```
public static ArrayList<TitleInfo> getCandidates(String fileName,
    StringBuffer content) {
    IRtfGroup rtfStructure = ParseRtf(fileName); // 获取Rtf 结构

    RtfInterpreterListenerDocumentBuilder docBuilder =
        new RtfInterpreterListenerDocumentBuilder(); //初始化
    RtfInterpreterListenerLogger interpreterLogger = null;
```



```

RtfInterpreterTool.Interpret(rtfStructure, interpreterLogger, docBuilder);
RtfDocument doc = (RtfDocument) docBuilder.getDocument();

if (doc == null) //读取失败
    return null;
ArrayList<IRtfVisual> rtfVisuals = doc.getVisualContent();

int maxFontSize = 0;
int maxPosition = 0;
for (IRtfVisual rtfv : rtfVisuals) { //找最大字体
    if (rtfv.getKind() == RtfVisualKind.Text) {
        int currentFontSize = ((IRtfVisualText) rtfv).getFormat().getFontSize();
        maxFontSize = Math.max(currentFontSize, maxFontSize);
    }
}

ArrayList<TitleInfo> candidates = new ArrayList<TitleInfo>();
for (int i = 0; i < rtfVisuals.size(); i++) {
    IRtfVisual rtfv = rtfVisuals.get(i);

    if (RtfVisualKind.Text == rtfv.getKind()) //只有 Text 才有文字属性
    {
        //换行前以第一种字符格式作为整行格式, 除了字体大小。
        String fontName = ((IRtfVisualText) (rtfv)).getFormat()
            .getFont().getName(); //首字符字体
        Color tc = ((IRtfVisualText) (rtfv)).getFormat()
            .getForegroundColor().getAsDrawingColor();
        boolean isBold = ((IRtfVisualText) (rtfv)).getFormat()
            .getIsBold();
        RtfTextAlignment alignment = ((IRtfVisualText) (rtfv))
            .getFormat().getAlignment();
        int fontSize = ((IRtfVisualText) (rtfv)).getFormat()
            .getFontSize();

        String oneRow = "";
        while (RtfVisualKind.Break != rtfv.getKind()) { //换行前的部分是一个整体
            if (rtfv.getKind() == RtfVisualKind.Text)
                oneRow += ((IRtfVisualText) rtfv).getText();
            i++;
            rtfv = rtfVisuals.get(i);
        }
    }
}

```

```

        //公文属性判断
        if (isProSendGovUnit(oneRow, tc) || //发文单位
            isProReceiGovUnit(oneRow, tc) || //收文单位
            isFileNum(oneRow)) { //文号
            maxPosition++;
            continue;
        }

        if (RtfTextAlignment.Center == alignment
            || fontSize == maxFontSize) { //居中、最大字体加入候选标题
            candidates.add(
                new TitleInfo(oneRow,
                               fontSize,
                               maxPosition,
                               fontName,
                               isBold));
        } else
            //不居中的是正文内容
            content.append(oneRow + " ");
    }
    maxPosition++;
}

return candidates;
}

```

对候选标题评分的代码如下所示。

```

public static void rankTitle(ArrayList<TitleInfo> titles, String content) {
    HashSet<String> stopWords = StopSet.getInstance(); //停用词表
    HashMap<String, Double> contentWords = new HashMap<String, Double>();

    int maxLength = 0;
    int maxFontSize = 0;
    int width = 440; //正文宽度
    int firstPosition = 9999;

    for (int i = 0, j = 1; j < titles.size(); i++, j++) { //第一步: 合并可能的标题
        TitleInfo ti = titles.get(i);
        TitleInfo tj = titles.get(j);
        firstPosition = Math.min(firstPosition, ti.position);
    }
}

```

```

if (ti.fontSize == tj.fontSize //字体大小一致
    && ti.position + 1 == tj.position //上下连贯
    && titles.get(i).text.length() > 1
    && titles.get(j).text.length() > 1) {
    if (!(tj.text.startsWith(" "))) {
        TitleInfo tTmp = ti;
        tj.mergeTo = i;
        while (tTmp.mergeTo != -1) {
            tj.mergeTo = tTmp.mergeTo;
            tTmp = titles.get(tTmp.mergeTo);
        }
        //向前合并标题同时删除
        titles.get(tj.mergeTo).text = titles.get(tj.mergeTo).text
            .trim()
            + tj.text.trim();
        tj.text = " ";
        if (ti.fontSize * ti.text.length() > width) { //因为字符过多而换行
            width *= 2;
            titles.get(tj.mergeTo).weight += 0.2;
        }
    }
}

for (TitleInfo m : titles) { //第二步: 分词, 确定候选标题的长度与位置范围
    if (m.text.trim().length() >= 2) { //非空标题分词
        maxLength = Math.max(maxLength, m.text.length());
        maxFontSize = Math.max(maxFontSize, m.fontSize);

        //标题分词, 建向量
        ArrayList<CnToken> taggedTitle = Tagger.getFormatSegResult(m.text);

        m.words = new HashMap<String, Double>();
        for (CnToken ct : taggedTitle) {
            if ("m".equals(ct.type()) || "t".equals(ct.type())
                || stopWords.contains(ct.termText()))
                continue; //去除停用词

            Double val = m.words.get(ct.termText());
            if (val != null) {
                m.words.put(ct.termText(), new Double(val + 1.0));
            }
        }
    }
}

```



```

        } else
            m.words.put(ct.termText(), new Double(1.0));
    }
    m.position -= firstPosition;
}
//对正文分词, 建向量
ArrayList<CnToken> taggedContent = Tagger.getFormatSegResult(content);
for (CnToken ct : taggedContent) {
    if ("w".equals(ct.type()) || "m".equals(ct.type())
        || "t".equals(ct.type())
        || stopWords.contains(ct.termText()))
        continue;//去除停用词

    if (contentWords.containsKey(ct.termText())) {
        contentWords.put(ct.termText(), new Double(contentWords.get(ct
            .termText()) + 1));
    } else
        contentWords.put(ct.termText(), new Double(1.0));
}
double contentNorm = calculateNorm(contentWords);

for (int i = 0; i < titles.size(); i++){//第三步: 综合评价候选标题权重
    TitleInfo t = titles.get(i);
    if (t.text.trim().length() >= 2) {
        double lengthWeight = getLengthWeight(t.text.length());
        double fontSizeWeight = getFontSizeWeight(t.fontSize,
            maxFontSize);
        double positionWeight = getPositionWeight(t.position);
        //计算可选标题与全文的相似度, 用夹角余弦来衡量相似度
        double semanticWeight = getSimilarity(t.words, contentWords,
            contentNorm);

        //计算综合权重
        Double compositiveWeight = new Double(t.weight
            * Math.pow(lengthWeight * fontSizeWeight
                * positionWeight, 1.0 / 3) * semanticWeight);
        //得出标题的最终权重
        if (t.isBold)
            t.weight = compositiveWeight * 1.1;
        else

```

```

        t.weight = compositiveWeight;
    }
}

```

最后简单地取最大分值对应的标题。

```

public static String getBestTitle(ArrayList<TitleInfo> titles) {
    double max = 0; //记录最大分值
    String bestTitle = null; //记录最好标题
    for (TitleInfo t : titles) {
        if (t.weight > max && t.text.trim().length() >= 2) {
            max = t.weight;
            bestTitle = t.text;
        }
    }
    return bestTitle;
}

```

### 5.6.5 从 Excel 文件中提取标题

为了提取 Excel 文件的标题，首先从每个工作表找最可能的标题，然后从多个工作表中再次挑选最可能的标题。

定义封装单元格属性的类，其中包含了用来计算标题重要度的一些属性。

```

public class CellInfo{
    public String text;//文本内容
    public short fontSize;//字体大小
    public short alignment;//对齐方式
    public boolean boldness;//是否黑体
    public int rowPos;//所在行的位置
    public double weight;//重要度
    public boolean isUnique;//独立成行
    public CellInfo(String t, short fs, int rp, short align, boolean bold){
        text = t;
        fontSize = fs;
        rowPos = rp;
        alignment = align;
        boldness = bold;
    }
}

```

```

        weight = 1.0;
        isUnique = false;
    }
}

```

通过遍历工作表中的每个字符型单元格来取得每个工作表的最好标题。

```

private TitleInf getSheetBestTitle(HSSFSSheet sheet, HSSFWorkbook wb){
    Iterator<HSSFRow> riter = sheet.rowIterator();//按行遍历工作表
    ArrayList<CellInfo> titles = new ArrayList<CellInfo>();

    int maxFontSize = 0;
    int rowCount = 0;
    while (riter.hasNext())    {
        rowCount ++;
        int columnCount = 0;
        HSSFRow row = (HSSFRow) riter.next();//按行遍历
        Iterator<HSSFCell> citer = row.cellIterator();

        while(citer.hasNext()) {
            HSSFCell cell = citer.next();//每行再按列遍历
            int cellType = cell.getCellType();
            HSSFCellStyle cellStyle = cell.getCellStyle();

            if (cellType != HSSFCell.CELL_TYPE_BLANK ) { //非空
                columnCount ++;
            }

            if (cellType == HSSFCell.CELL_TYPE_STRING) { //字符型
                String cellString = cell.toString().trim();//取得单元格内的文本
                if (cellString.length() >= 2)    {
                    HSSFFont cellFont = cell.getCellStyle().getFont(wb);

                    short fontheight = cellFont.getFontHeight();
                    short al = cellStyle.getAlignment();
                    short boldness = cellFont.getBoldweight() ;
                    maxFontSize = Math.max(maxFontSize, (int) fontheight);
                    CellInfo ci = new CellInfo(cellString,
                                                fontheight,
                                                rowCount,
                                                al,
                                                boldness == HSSFFont.BOLDWEIGHT_BOLD);

                    titles.add(ci);
                }
            }
        }
    }
}

```



```

    }
}
}
if (columnCount == 1) //这行只有这个
    titles.get(titles.size() - 1).isUnique = true;
}

if (titles.size() == 0)
    return new TitleInf("", 0);
else
    return selectBestTitle(titles, maxFontSize, rowCount);
}

```

标题类包含了标题的文本内容和重要度。

```

public class TitleInf{
    public String text;//文本内容
    public double weight;//重要度
    public TitleInf(String t, double w){
        text = t;
        weight = w;
    }
}

```

取得整个 Excel 文件最可能的标题。

```

public String getTitle(String fileName) throws Exception{
    InputStream is = new FileInputStream(fileName);
    HSSFWorkbook wb = new HSSFWorkbook(new POIFSFileSystem(is));
    ArrayList<TitleInf> candidate = new ArrayList<TitleInf>();//候选标题
    int activeSheetIndex = wb.getActiveSheetIndex();//取得活跃工作表的编号

    int sheetsNum = wb.getNumberOfSheets();
    for (int i = 0 ; i < sheetsNum ; i++){//取得每个工作表最可能的标题
        TitleInf bti = getSheetBestTitle(wb.getSheetAt(i), wb);
        if (i == activeSheetIndex)
            bti.weight *= 3.0;
        candidate.add(bti);
    }

    //取得评分最高的候选标题
    double maxWeight = 0;
}

```

```

String bestTitle = null;
for (TitleInf curTitle : candidate) {
    if (curTitle.weight >= maxWeight){
        maxWeight = curTitle.weight;
        bestTitle = curTitle.text;
    }
}

is.close();
return bestTitle;
}

```

### 5.6.6 从 PowerPoint 文件中提取标题

PowerPoint 文件由一个或多个幻灯片（Slide）组成。第一张幻灯片的标题往往是整个 PowerPoint 文件的标题，提取标题实现代码如下所示。

```

SlideShow ss = new SlideShow(new HSLFSlideShow(is)); // is 是 PPT 文件的输入流
Slide[] slides = ss.getSlides(); // 获得每一张幻灯片
return slides[0].getTitle(); // 返回第一张幻灯片的标题

```

## 5.7 图像的 OCR 识别

抓取过程中，如果碰到包含价格或联系方式等信息的图片，需要转换成文字信息。例如，以图片表示的价格：**¥ 1119.00**。在搜索引擎中实现图片或视频搜索时，也可以把图片或视频中的文字信息提取出来，然后按文字来查找。

从图片中识别出字符叫作光学字符识别（Optical Character Recognition），简称 OCR。OCR 过程的核心是切割出包含单个字符的图像，然后把图像分类到一个具体的字符。可以把识别出单个文字看成一个分类的问题。需要提取出能作为分类依据的特征，去掉一些和分类无关的特征。

自己写个训练集，训练出支持向量机（SVM）分类模型。每个字对应一个图片文件训练，然后对要识别的图像分割，对分割后的图像分类。

OCR 流程如下所示。

(1) 对要识别的图像区域进行二值化处理，识别出字体和背景的颜色。如果有必要，还需要对图像中的干扰去噪。

(2) 对二值化处理后的图像切分。图像切分就好像裁缝裁布一样，一般从空白处切开。因为字号变化导致对应的字符图像尺寸相差可能达到数十倍，所以还需要把切割后的图像大小归一化。

(3) 判断包含单个字符的图像应该识别成哪个字符，这是个分类的问题，可以采用机器学习的方法实现，是有监督的学习（supervised learning）。先要准备好包含对应字符的图像样本及应该识别出的字符答案作为训练库。事先训练出模型后，识别时加载用分类方法学习出的分类模型；分类方法可以选择 SVM 或者最大熵分类等。在 OCR 识别过程中，需要对从图像提取出的数据集进行分类，因此合适的分类器对结果的预测起着非常重要的作用。近年来，支持向量机作为一种基于核方法的机器学习技术，不仅有强烈的理论基础而且有成功经验。这里采用开源的 SVM 软件 LIBSVM 来对图像分类。

(4) 执行分类并输出结果。如果有必要可以根据上下文猜测最有可能的输出结果来降低识别错误。

在学习阶段，OCR 类从单个字符图片中学习出分类模型，然后在识别阶段加载分类模型。调用图像识别模块接口的代码如下所示。

```
OCR ocr = new OCR();
ocr.preload(); //加载模型
String imgFile = "D:/ocr/sample-images/ESfjydz.png";
String text = ocr.recognize(imgFile); //识别的过程
System.out.println("\nresult:"+text);
```

### 5.7.1 读入图像

首先要支持读入多种常见图片文件格式，例如 jpg、gif 等。Java 中进行图像 I/O 有一些方法，其中的 Image I/O API 支持读写常见图片格式。Image I/O API 提供了加载和保存图片的方法，支持读取 GIF、JPEG 和 PNG 图像，也支持写 JPEG 和 PNG 图像，但是不支持写 GIF 文件。Java Image I/O API 主要在 javax.imageio 包。JDK 已经内置了常见图片格式的插件，但它提供了插件体系结构，第三方也可以开发插件支持其他图片格式。



javax.imageio.ImageIO 类提供了一组静态方法进行最简单的图像 I/O 操作。读取一个标准格式（GIF、PNG 或 JPEG）的图片很简单。

```
File f = new File("c:\\images\\myimage.gif");
BufferedImage bi = ImageIO.read(f);
```

Java Image I/O API 会自动探测图片的格式并调用对应的插件进行解码，当安装了一个新插件，新的格式会被自动理解，程序代码不需要改变。

## 5.7.2 准备训练集

训练集就是一些图片对应的文字。新建一个空白图，然后写一个字到图片文件，可以把字作为这个图片的名字。这样就得到字对应的图像点阵，也就是位图。例如，生成“北”这个字对应的位图。

```
public static void writeImg(String imageFileName,String charToImg) throws Exception {
    int width = Entry.DOWNSAMPLE_WIDTH*5;
    int height = Entry.DOWNSAMPLE_HEIGHT*5;
    BufferedImage tag = new BufferedImage(width / 1, height / 1,
        BufferedImage.TYPE_INT_RGB); //构造 Image 对象
    Graphics g = tag.getGraphics();
    g.drawString(charToImg, 10, 30); //写指定的字符
    File file = new File(imageFileName); //输出到文件流
    ImageIO.write(tag, "jpg", file);
}

public static void main(String[] args) throws Exception {
    ReadImage.writeImg("D:/北.jpg","北");
}
```

宽度是 12，高度是 18 的二值图像区域对应一个字符。把训练集写入配置文件。

```
b:11000000000011000000000011000000000011000000000011001111110011001111110011110000001111
1100000011110000000011110000000011110000000011110000000011110000000011110000000011111100
000011111100000011110011111100110011111100
c:00011111111100011111111000111111111100000000011100000000011100000000011100000000011
1000000000111000000000111000000000111000000000111000000000111000000000111000
0000001110000000000000111111111000111111111
```

[illegible]

因为这样的训练集不太直观，所以把训练集导出成二值图片。

```
String train =
"11000000000011000000000011000000000011000000000110011111100110011111100111100000011111
1000000111100000000111100000000111100000000111100000000111100000000111100000000111111000
000111111000000011110011111100110011111100";

BufferedImage binarized = new BufferedImage(Entry.DOWNSAMPLE_WIDTH,
        Entry.DOWNSAMPLE_HEIGHT, BufferedImage.TYPE_INT_RGB);

File file = new File("d:/train.bmp");

int pixels[] = new int[Entry.DOWNSAMPLE_WIDTH * Entry.DOWNSAMPLE_HEIGHT];

for (int i = 0; i < train.length(); ++i) {
    char c = train.charAt(i);
    if (c == '0') {
        pixels[i] = getWhite();
    }else{
        pixels[i] = getBlack();
    }
}

//设置 scansize 的值成为图片的宽度
binarized.setRGB(0, 0, Entry.DOWNSAMPLE_WIDTH, Entry.DOWNSAMPLE_HEIGHT, pixels, 0,
Entry.DOWNSAMPLE_WIDTH);

ImageIO.write(binarized, "bmp", file);
```

### 5.7.3 图像二值化

包含笔画的区域叫作前景区域，不包含笔画的区域叫作背景区域。通过二值化区分前景区域和背景区域。例如，b 的图像区域变成一个 0 和 1 组成的图像，如图 5-13 所示。

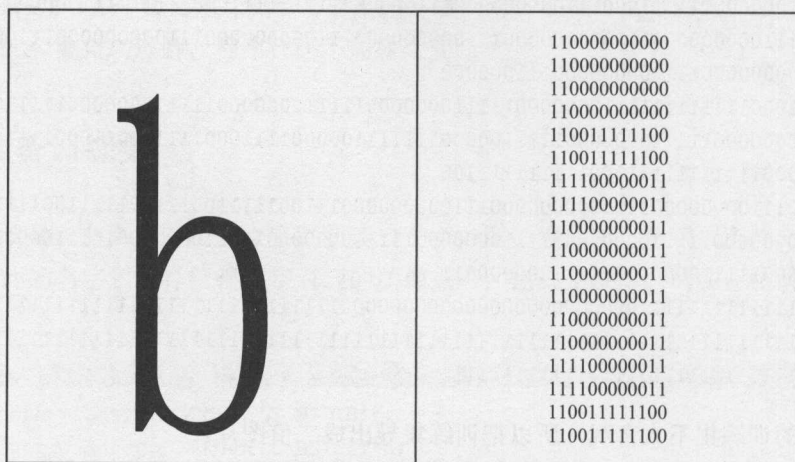


图 5-13 字符 b 以及它的 0 和 1 组成的二值化结果

对要识别的图像区域进行二值化处理是很重要的步骤。在实际应用过程中如果图像二值化处理不好，那么识别出来的可能都是 1 了，也就是说图片需要预处理成 0101 这样的黑白格式。

有些待识别的图片存在干扰，可以考虑按颜色分辨出干扰。例如，有的图片只有三种颜色，白色背景、红色字、黑色水平线。黑色水平线是干扰，所以把黑色去掉，只留下白色背景和红色字。

人类的视网膜包含三种感光锥形细胞，它们按照马赛克的方式排列，分别对红色、绿色和蓝色光敏感。人看到的各种颜色是由视网膜上三种感觉神经发出的信号组合而成。一幅彩色图像的像素矩阵中每个像素由 RGB 三种颜色按一定的比例混合形成一种颜色来表示，比如纯黑是因为图像上没有任何色光存在，相当于 RGB 三种色光都没有发光，所以黑色是 RGB(0,0,0)。纯白是 RGB 三种色光都发到最强的亮度，所以纯白是 RGB(255,255,255)。纯红色是 RGB(255,0,0)。

```
BufferedImage bi = ImageIO.read(f);
```

```
//得到图片的宽和高
```



```

int width = bi.getWidth(null);
int height = bi.getHeight(null);

//读取像素
int pixels[] = new int[width * height];
bi.getRGB(0, 0, width, height, pixels, 0, width);
int pixel = pixels[0];
int r = (pixel >> 16) & 0xff; //高8位
int g = (pixel >> 8) & 0xff; //中间8位
int b = (pixel) & 0xff; //低8位

```

将彩色图片黑白化的常规做法是先转化为灰度图，然后把灰度图转化为黑白图。为了用计算机来表示和处理颜色，必须采用定量的方法来描述颜色，即建立颜色模型。常用的有红、绿、蓝为原色的 RGB 颜色模型和描述色彩饱和度的 HSV 颜色模型。RGB 彩色包含了亮度信息，即 YUV 模型中的 Y 分量，灰度计算公式如下所示。

$$\text{gray} = 0.229 \times r + 0.587 \times g + 0.114 \times b$$

为了归一化，这里的  $0.229 + 0.587 + 0.114 = 1$ 。

计算一个 RGB 颜色模型表示的像素的灰度。

```

/**
 * 计算一个 RGB 颜色的灰度
 * @param pixel 8 位 RGB 颜色
 * @return 对应的灰度
 */
private static int getGray(int pixel) {
    int r = (pixel >> 16) & 0xff; //高8位
    int g = (pixel >> 8) & 0xff; //中间8位
    int b = (pixel) & 0xff; //低8位

    //按公式计算出灰度值
    int gray = (int) (0.229 * r + 0.587 * g + 0.114 * b);
    return gray;
}

```

可以根据灰度值来确定像素的二值化结果。最简单的方式是固定阈值法。例如，若灰度大于 128，则认为是白色，灰度小于 128，则认为黑色。可以简单的根据灰度值用固定阈值法转换成黑白图片。

```

BufferedImage img = ImageIO.read(file); //读入彩色图
int h = img.getHeight();
int w = img.getWidth();
int[] pixels = new int[w * h];
//PixelGrabber 类实现了 ImageConsumer 接口
//它可以连接到一个图像或 ImageProducer 对象来检索该图像中的像素子集
//可以通过 PixelGrabber 类获得 Image 对象的像素
PixelGrabber pg = new PixelGrabber(img, 0, 0, w, h, pixels, 0, w);
pg.grabPixels();
for (int i = 0; i < h; i++) {
    for (int j = 0; j < w; j++) {
        int rgb = pixels[w * i + j];
        int gray = getGray(rgb);
        if (gray > 200) { //根据固定阈值设置成黑色或者白色
            alpha = 255;red = 255;green = 255;blue = 255;gray = 255; //白色
        }
        if (gray < 200) {
            alpha = 0;red = 0;green = 0;blue = 0;gray = 0; //黑色
        }
        int color = (alpha << 24) | (gray << 16) | (gray << 8) | gray;
        pixels[w * i + j] = color;
        img.setRGB(j, i, color);
    }
}
ImageIO.write(img, "png", new File(outFile)); //写出黑白图

```

可以用阈值法转换灰度图像为二值图像。OTSU 算法是自适应计算全局阈值的简单高效方法。当取最佳阈值时，背景应该与前景差别最大，关键在于如何选择衡量差别的标准。

二值化双峰法的原理是：图像由前景和背景组成，在灰度直方图上，前后二景都形成高峰，在双峰之间的最低谷处就是图像的阈值所在，可根据此值，对图像进行二值化。

统计图片的灰度直方图的代码。

```

int grayValues[]; //每个像素对应的灰度值

BufferedImage bi = readImageFromFile(imageFile);

//得到图片的宽和高
int width = bi.getWidth(null);

```

```

int height = bi.getHeight(null);

//读取像素
int pixels[] = new int[width * height];
bi.getRGB(0, 0, width, height, pixels, 0, width);

//计算每个像素的灰度, 保存下来
grayValues = new int[width * height];
for(int i = 0; i < width * height; i++) {
    grayValues[i] = getGray(pixels[i]);
}

int hist[] = new int[256]; //存放每种灰度的出现次数

for(int i = 0; i < grayValues.length; i++) {
    hist[grayValues[i]]++; //统计每种灰度出现的次数, 即得到直方图
}

```

OTSU 算法采用最大类间方差衡量差别, 所以 OTSU 算法也称为最大类间差法。

记  $t$  为前景与背景的分割阈值, 前景点数占图像比例为  $w_0$ , 平均灰度为  $u_0$ ; 背景点数占图像比例为  $w_1$ , 平均灰度为  $u_1$ , 则图像的总平均灰度为:

$$u = w_0 \times u_0 + w_1 \times u_1$$

前景和背景图像的方差公式为:

$$g = w_0 \times (u_0 - u) \times (u_0 - u) + w_1 \times (u_1 - u) \times (u_1 - u)$$

$$= w_0 \times w_1 \times (u_0 - u_1) \times (u_0 - u_1)$$

寻找能让方差最大的阈值  $t$ 。

```

//使用 OTSU 算法得到二值化的阈值
private static int otsuThreshold(BufferedImage original) {
    int[] histogram = imageHistogram(original);
    int total = original.getHeight() * original.getWidth();

    float sum = 0;
    for(int i=0; i<256; i++) sum += i * histogram[i];

    float sumB = 0;

```



```

int wB = 0;
int wF = 0;

float varMax = 0;
int threshold = 0;

for(int i=0 ; i<256 ; i++) {
    wB += histogram[i];
    if(wB == 0) continue;
    wF = total - wB;

    if(wF == 0) break;

    sumB += (float) (i * histogram[i]);
    float mB = sumB / wB;
    float mF = (sum - sumB) / wF;

    float varBetween = (float) wB * (float) wF * (mB - mF) * (mB - mF);

    if(varBetween > varMax) {
        varMax = varBetween;
        threshold = i;
    }
}

return threshold;
}

```

二值化后的图片不能存成 jpg 格式，因为 jpg 格式是有损压缩，应该把二值化后的图片存成 bmp 格式。

```

public static void main(String[] args) throws IOException {
    String fileName = "D:/search/passCode2";
    File original_f = new File(fileName + ".jpg");
    String output_f = fileName + "_bin";
    BufferedImage original = ImageIO.read(original_f);
    BufferedImage grayscale = toGray(original); //灰度化
    BufferedImage binarized = binarize(grayscale); //二值化
    writeImage(binarized, output_f); //存成 bmp 格式
}

```

```
private static void writeImage(BufferedImage binarized,String output) throws IOException {
    File file = new File(output + ".bmp");
    ImageIO.write(binarized, "bmp", file); //存成 bmp 格式的图片
}
```

一般而言, 写字不可能写到四个角的像素上去, 因此用这 4 个点的颜色作为背景颜色。

自适应阈值根据当地图像特征为每个像素都设置一个阈值, 而全局阈值的方法为所有的像素设置统一的一个阈值。因此, 在某些情况下, 如照明条件不好的情况下, 自适应阈值提供了较好的效果。

BradleyLocalThresholding 实现了自适应阈值的图像二值化, 可以从 <http://www.aforgenet.com/projects/iplab/>网上下载二值化的软件。

## 5.7.4 切分图像

对字符区域定位的 CharRange 类把带有字符的区域从图像中分离出一个矩形框出来。

```
public class CharRange {
    int x; //横坐标位置
    int y; //纵坐标位置
    int width; //宽度
    int height; //高度
}
```

Entry 类对图片进行垂直和水平方向的扫描, 实现了切割字符并且把字符大小归一化, 也就是统一图像宽度和高度。Entry 类的主要成员变量及方法如下所示。

```
public class Entry {
    static final int DOWNSAMPLE_WIDTH = 12; //样本数据宽度
    static final int DOWNSAMPLE_HEIGHT = 18; //样本数据高度
    protected Image entryImage; //存储检测的图像
    protected Graphics entryGraphics; //处理图形图像
    protected int pixelMap[]; //存储图像像素

    //水平扫描图像并进行像素检测
    protected boolean hLineClear(int x, int w, int y) {
        int totalWidth = entryImage.getWidth(null);
        for (int i = x; i <= w; i++) {
```

```

        if (pixelMap[(y * totalWidth) + i] != -1)
            return false;
    }
    return true;
}

//垂直扫描图像并进行像素检测
protected boolean vLineClear(int x) {
    int w = entryImage.getWidth(null);
    int h = entryImage.getHeight(null);
    for (int i = 0; i < h; i++) {
        if (pixelMap[(i * w) + x] != -1)
            return false;
    }
    return true;
}

//找到水平扫描时的上边界和下边界
void findVBound(CharRange cr) {
    for (int i = 0; i < cr.height; i++) {
        if (!hLineClear(cr.x, cr.width, i)) {
            cr.y = i;
            break;
        }
    }
    for (int i = cr.height - 1; i >= 0; i--) {
        if (!hLineClear(cr.x, cr.width, i)) {
            cr.height = i;
            break;
        }
    }
}

//找到垂直扫描时的左边界和右边界
protected ArrayList<CharRange> findHBounds(int w, int h) {
    ArrayList<CharRange> bounds = new ArrayList<CharRange>();
    int begin = 0;
    int end = w;
    boolean lastState = false;
    boolean curState = false;
    for (int i = 0; i < w; i++) {

```



```

        if (vLineClear(i)) {
            System.out.println("find blank:" + i);
            curState = false;
        } else {
            curState = true;
        }
        if (!lastState && curState) {
            begin = i;
        } else if (lastState && !curState) {
            end = (i - 1);
            CharRange cr = new CharRange(begin, 0, end, h);
            bounds.add(cr);
        }
        lastState = curState;
    }
    if (curState) {
        CharRange cr = new CharRange(begin, 0, w - 1, h);
        bounds.add(cr);
    }
    return bounds;
}

```

//发现图像边界

```

protected ArrayList<CharRange> findBounds(int w, int h) {
    ArrayList<CharRange> bounds = findHBounds(w, h);
    for (CharRange cr : bounds) {
        findVBound(cr);
    }
    return bounds;
}

```

//对样本数据进行采样、归一化

```

public ArrayList<SampleData> downSample() {
    int w = entryImage.getWidth(null);
    int h = entryImage.getHeight(null);
    ArrayList<SampleData> samples = new ArrayList<SampleData>();
    PixelGrabber grabber = new PixelGrabber(entryImage, 0, 0, w, h, true);
    grabber.grabPixels();
    pixelMap = (int[]) grabber.getPixels();
    ArrayList<CharRange> bounds = findBounds(w, h);
    for (CharRange cr : bounds) {

```

```

        SampleData data = new SampleData("?", DOWNSAMPLE_WIDTH,
            DOWNSAMPLE_HEIGHT);
        System.out.println(cr);
        double ratioX = (double) (cr.width - cr.x + 1)
            / (double) DOWNSAMPLE_WIDTH;
        double ratioY = (double) (cr.height - cr.y + 1)
            / (double) DOWNSAMPLE_HEIGHT;
        for (int y = 0; y < data.getHeight(); y++) {
            for (int x = 0; x < data.getWidth(); x++) {
                if (downSampleQuadrant(x, y, ratioX, ratioY, cr.x, cr.y))
                    data.setData(x, y, true);
                else
                    data.setData(x, y, false);
            }
        }
        data.ratio = (double) (cr.width - cr.x + 1)
            / (double) (cr.height - cr.y + 1);
        data.ratio = (data.ratio - 1) / 4;
        samples.add(data);
    }
    return samples;
}

```

//在样本数据的指定范围内进行像素扫描

```

protected boolean downSampleQuadrant(double x, double y, double ratioX,
    double ratioY, double downSampleLeft, double downSampleTop) {
    int w = entryImage.getWidth(null);
    int startX = (int) (downSampleLeft + (x * ratioX));
    int startY = (int) (downSampleTop + (y * ratioY));
    int endX = (int) (startX + ratioX);
    int endY = (int) (startY + ratioY);
    for (int yy = startY; yy <= endY; yy++) {
        for (int xx = startX; xx <= endX; xx++) {
            int loc = xx + (yy * w);
            if (pixelMap[loc] != -1)
                return true;
        }
    }
    return false;
}
}

```

如果无法找到边界，可以采用图像对齐的方法。

图像对齐就是匹配两幅图像：模板  $T$  和其他图像  $I$ 。图像对齐是一个迭代的最小化过程，该技术利用图像的灰度梯度信息迭代搜索两幅图像之间的最佳匹配。根据模版图像的作用，可以分为正向和反向算法。正向算法把模板图像  $T$  变换到图像  $I$ 。反向算法把图像  $I$  变换到图像  $T$ 。

### 5.7.5 SVM 分类

为了调用通用的模式识别代码来识别采样后的字符图像代表哪个字符，需要把这个图像识别问题抽象化为一般的分类问题。对于一个输入对象  $x$ ，有对应的输出类型  $y$ 。在这里，考虑对数字图片分类，输入是采样后的“0101”序列，输出则是 0~9 十个数字。“0101”序列中的每一位作为一个特征。训练集中的每个  $x_i$  都有对应的  $y_i$  对应的  $m$  个训练实例，记作  $(x_1; y_1); \dots; (x_m; y_m)$ 。每个  $x_i$  有  $k$  个特征。

SVM 的分类方法可以分为训练过程和识别过程：在训练过程中，要准备一些识别好的图片，学习出分类模型；在识别过程中，调用学习好的分类模型来分类。LIBSVM (<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>) 是台湾大学的林智仁教授开发的，它是可以解决分类问题的 SVM 软件包。LIBSVM 支持多种编程语言，这里采用 Java 版本。

LIBSVM 使用 `svm_train` 方法训练并返回一个支持向量机模型，然后可以调用 `svm_predict` 方法根据已训练好的模型进行预测。

LIBSVM 的输入是实数向量，输出特征是正整数表示的类型。

推荐使用  $m$  个数表示一个  $m$  类的属性。 $m$  个数字中，仅一个是一，其他都是零。例如，红、绿、蓝三种颜色的属性分类分别是(0,0,1)、(0,1,0)、(1,0,0)。

LIBSVM 的输入是特征组成的向量，这里用 0 和 1 表示一个采样点特征。假设只采样 4 个点，则对应向量(0,1,0,1)。

在 `svm_node` 类的实例中用稀疏的方式存储单个训练向量中的单个特征。

```
public class svm_node{
    public int index; //特征编号
    public double value; //特征对应的值
}
```



因为汉字是大字符集，识别较困难，一般以笔画为依据。对于价格和邮件地址等单纯由数字和字母组成的文本，可以采用简单的像素取样的方法。

在 `svm_problem` 类的实例中存储训练集中的所有样本及其所属类别。

```
public class svm_problem {
    //训练数据的实例数
    public int l;
    //存放它们的目标值，类型值用整型数据，回归值用实数
    public double[] y;
    //训练向量用二维矩阵表示
    //矩阵的第一个维度的长度是训练的实例数，第二个维度是特征数量
    public libsvm.svm_node[][] x;
}
```

训练数据的输入样本组成了一个 `SampleData` 实例的数组。

```
public class SampleData{
    protected boolean grid[][]; //用二维布尔数组存储的采样数据
    public double ratio; //宽高比，即 width/height
    protected String letter; //对应的字符或字符串
}
```

学习输入样本，得到分类模型。

```
//输入特征数量
int featureNum = Entry.DOWNSAMPLE_HEIGHT * Entry.DOWNSAMPLE_WIDTH + 1;
int outputNumber = sampleList.size();

//准备好训练集
TrainingSet set = new TrainingSet(featureNum, outputNumber);
set.setTrainingSetCount(sampleList.size());

for (int t = 0; t < sampleList.size(); t++) {
    int idx = 0;
    SampleData ds = (SampleData) sampleList.get(t);
    for (int y = 0; y < ds.getHeight(); y++) {
        for (int x = 0; x < ds.getWidth(); x++) {
            set.setInput(t, idx++, ds.getData(x, y) ? .5 : -.5);
        }
    }
    set.setInput(t, idx++, (ds.ratio));
    set.setOutput(t, t);
}
```

```

}

svm_parameter param = new svm_parameter();//设置模型参数
param.svm_type = svm_parameter.C_SVC;//标准算法
param.kernel_type = svm_parameter.RBF; //训练采用 RBF 核函数
param.degree = 3; //多项式核函数的阶次
param.gamma = 1.0 / featureNum;
param.coef0 = 0;
param.nu = 0.5;
param.cache_size = 100; //缓存内存大小设置为 100M
param.C = 1;
param.eps = 1e-3;
param.p = 0.1;
param.shrinking = 1; //使用启发式
param.probability = 0;
param.nr_weight = 0;
param.weight_label = new int[0];
param.weight = new double[0];

svm_problem prob = new svm_problem();//训练集

prob.l = outputNumber;
prob.x = new svm_node[prob.l][];
prob.y = new double[prob.l];
for (int i = 0; i < prob.l; i++){//设置输入及对应的输出
    prob.x[i] = set.getInputSet(i);
    prob.y[i] = set.getOutput(i);
}

//检查参数的有效性
String error_msg = svm.svm_check_parameter(prob, param);

//如果有错误则退出
if (error_msg != null) {
    System.err.print("Error: " + error_msg + "\n");
    System.exit(1);
}

//执行训练
model = svm.svm_train(prob, param);

```

执行对指定图片的分类过程。

```

LoadImage li = new LoadImage();//加载图像

Entry entry = new Entry();
entry.entryImage = li.loadImageFromFile(new File(imgFile));//加载图像

ArrayList<SampleData> samples = entry.downSample();//采样

//特征数组
svm_node[] input =
    new svm_node[Entry.DOWNSAMPLE_WIDTH* Entry.DOWNSAMPLE_HEIGHT + 1];

StringBuilder text = new StringBuilder();//存放识别结果

for (SampleData ds : samples) {//分别识别每个切分出的图像
    int idx = 0;
    for (int y = 0; y < ds.getHeight(); y++) {
        for (int x = 0; x < ds.getWidth(); x++) {
            input[idx] = new svm_node();//设置分类特征, 把采样点作为分类特征
            input[idx].index = idx + 1;
            input[idx].value = ds.getData(x, y) ? .5 : -.5;//设置分类特征值
            ++idx;
        }
    }

    input[idx] = new svm_node();
    input[idx].index = idx + 1;
    input[idx].value = ds.ratio;

    int best = (int) svm.svm_predict(model, input);//用 SVM 方法分类

    String result = map[best];//取得分类 Id 对应的字符串

    //对全黑图像字符的特殊处理
    if (result.equals("1") || result.equals(".") || result.equals("-")) {
        if (ds.ratio == 0) {
            result = ".";
        } else if (ds.ratio < 0) {
            result = "1";
        } else if (ds.ratio > 0) {
            result = "-";
        }
    }
}

```



```
text.append(result);
```

尽管 SVM 分类方法返回的结果准确度已经很高，但是仍然可能把字母“o”识别成了数字“0”，图像过于相似的情况下要靠上下文来识别，比如后面是 h，则前面的符号更可能是字母“o”而不是数字“0”，实现上可以参考 HMM（隐马模型）的介绍。

### 5.7.6 识别汉字

GB2312 中定义了 6 763 个汉字，直接使用常规的分类方法区分这么多字符有难度。中文字符的大小采用  $24 \times 24$  像素，也可以采用  $32 \times 32$  像素。

如果样本和要识别的图片都执行了二值化，则可以使用海明距离比较二进制位。如果海明距离差别位数少，则可以使用 SimHash 所用的方法找到相似的。

逐个比较样本和识别图的相似性速度太慢，可以用图像矩快速消除差别极大的字，然后逐个比较少量样本和识别图。具体来说可以用图像矩衡量一个字的轻重，比如“一”这个字很轻，而“壹”这个字很重。如果用 BitSet 存储图像二值化后的二进制位，则可以调用 BitSet.cardinality() 方法。

在图像上计算不变矩是对图像识别有用的属性。这里计算图像的几何矩。如果把图像看成是一块质量密度不均匀的薄板，其图像的灰度分布函数  $f(x, y)$  就是薄板的密度分布函数，则其各阶矩有着不同的含义，如零阶矩表示它的总质量；一阶矩表示它的质心；二阶矩又叫惯性矩，表示图像的大小和方向。

有的字笔画很多，看起来很重。有的字笔画少，看起来很轻。可以用零阶矩来度量字的轻重。“下”这个字的质心在右上方。“上”这个字的质心在右下方。可以用一阶矩来度量字的质心所在。

计算一个区域的零阶矩的代码如下所示。

```
double m00 = 0.0; //存放零阶矩的值
//计算零阶矩
for (int y=r.y; y<(r.y+r.height); y++) {
    for (int x=r.x; x<(r.x+r.width); x++) {
        currentPixel=ip.getPixelValue(x,y);
        m00+=currentPixel;
    }
}
```

```

}
}

```

一阶矩用一个数字表示从(0,0)开始的质心所在的位置。

考虑根据笔画的走势特征识别中文偏旁部首和整个文字。汉字分成偏旁部首，提取笔画特征。笔画从哪里开始，比如规定从左上角开始。然后接下来怎么走，类似二元连接那样，用一些二元关系的点来描述整个笔画，描述下一个点和上一个点的关系。

一个汉字包含很多个点，间隔若干个点取一个标志点，但是要把所有的转折点都包括进来。两个标志点中再取中间点，然后检查三点是否趋于直线，如果是就忽略，如果三点给出的线过于弯曲，或者角度很明显，就算作一个转折存起来，作为这两个点的属性记录下。

对点聚类，也就是把连通的点聚成一类。一个点的八邻域指最左、左上、最上、上右、最右、右下、最下、左下。首先找出一个点的所有邻接点，然后找邻接点的邻接点，依此类推，这就是八邻域聚类算法。判断此点八邻域中的最左、左上、最上、上右点的情况。如果都没有点，则表示一个新的类别的开始。如果此点八邻域中的最左和上右都有点，则标记此点为这两个中的最小的标记点，并修改大标记为小标记。如果此点八邻域中的左上和上右都有点，则标记此点为这两个中的最小的标记点，并修改大标记为小标记。否则按照最左、左上、最上、上右的顺序，标记此点为四个中的一个。

更快的方法是使用轮廓跟踪技术的线性时间分量标记算法（A Linear-Time Component-Labeling Algorithm Using Contour Tracing Technique），即使单字的识别率很高，但是如果只是简单地取最有可能的单字作为最终的识别率，则整个句子总的识别率仍然不会高，所以需要根据语言模型做后处理。

北海道札幌市

北—海—道—札—幌—市  
 此—侮—通—扎—中  
 每

一些汉字的笔画数量在十二笔以上。对于笔画多的汉字，需要提取一些局部特征。

## 5.7.7 训练 OCR

确保每个字符的样本数量至少有 10 个, 例如有的字存在笔迹粘连问题。对于生僻字来说, 只有 5 个样本也没问题。

### 1. 训练所需的准备

(1) 下载并安装 3.01 版本的 `tesseract`。如果下载的是压缩包版, 解压即可, 解压到 `E:\Tesseract-ocr` 目录。

(2) 下载并安装 `jTessBoxEditor` 工具, 它是一个盒子文件编辑器, 用来编辑训练文件, 直接下载地址在 <http://vietocr.sourceforge.net/training.html> 上。这个软件是用 Java 写的, 运行需要安装 `jre`, `jre` 比 `.net` 好装多了, 运行可以参看它的 `readme` 文件。

(3) 一张用来训练的 `tiff` 格式图片。

在不通过训练的前提下, 使用 `tesseract` 来识别一个订单号的内容, 发现错误率很高, 希望通过训练来提高准确率。

### 2. 训练过程

(1) 把 10 张图片合并为一张 `tiff` 格式的图片, 如何合并呢? 通过 `jTessBoxEditor` 的 `Merge Tiff` 来完成, 不过它的小缺点就是只能合并多张 `tiff` 格式的图片, 如果图片是 `jpg` 格式的, 需要先转换, 生成后的 `tiff` 图片叫作 `orderNo.tif`。

(2) 生成盒子文件。在 `orderNo.tif` 目录下打开一个命令行, 输入命令: `E:\Tesseract-ocr\tesseract.exe orderNo.tif orderNo batch.nochop makebox` 来生成一个盒子文件, 该文件记录了 `tesseract` 识别出来的每一个字及其位置坐标。

(3) 使用 `jTessBoxEditor` 打开 `orderNo.tif` 文件, 需要记住的是第 2 步生成的 `orderNo.box` 要和 `orderNo.tif` 文件同在一个目录下。逐个校正文字, 后保存。

(4) 运行 `Tesseract` 训练过程。输入命令: `E:\Tesseract-ocr\tesseract.exe orderNo.tif orderNo nobatch box.train`。

(5) 计算字符集。输入命令: `E:\Tesseract-ocr\unicharset_extractor.exe orderNo.box`。

(6) 新建文件 “`font_properties`”。如果是 3.01 版本, 那么需要在目录下新建一个名为 “`font_properties`” 的文件, 并且输入文本: `orderNo 0 0 0 0 0`, 意思是 `orderNo` 语言的字体为普通字体。并执行命令: `E:\Tesseract-ocr\mftraining.exe -F font_properties -U unicharset orderNo.tr`。



(7) 聚类, 输入命令: `E:\Tesseract-ocr\cntraining.exe orderNo.tr`。此时, 在目录下应该生成若干个文件, 把 `unicharset`、`inttemp`、`normproto`、`pfmtable` 这四个文件加上前缀 “`orderNo.`”, 然后输入命令: `E:\Tesseract-ocr\combine_tessdata.exe orderNo`, 会显示一个如下所示的结果。

```
Combining tessdata files
TessdataManager combined tesseract data files.
Offset for type 0 is -1
Offset for type 1 is 108
Offset for type 2 is -1
Offset for type 3 is 1660
Offset for type 4 is 327545
Offset for type 5 is 327781
Offset for type 6 is -1
Offset for type 7 is -1
Offset for type 8 is -1
Offset for type 9 is -1
Offset for type 10 is -1
Offset for type 11 is -1
Offset for type 12 is -1
```

必须确定第 2、4、5、6 行的数据不是 -1, 那么一个新的字典就算生成了。

此时, 目录下 “`orderNo.traineddata`” 的文件复制到了 `tesseract` 程序目录下的 “`tessdata`” 目录。

以后就可以使用该字典来识别了, 例如, `tesseract.exe test.jpg result -l orderNo`, 通过训练出来的新语言, 识别率提高了不少。

### 5.7.8 检测行

一个图片扫描之后按照文字的方向自动调整图片。如何识别行的方向?

首先使用 RLS (Recursive Least Squares) 自适应滤波器处理图像, 然后通过哈夫变换查找直线。根据点线的对偶性, 当给定图像空间的一些边缘点, 就可以通过哈夫变换确定连接这些点的直线方程。

运行长度平滑算法 (RLSA) 可用于块分割和文本的消除歧义。为文档分析系统开发的这个方法包括两个步骤。首先, 一个分割过程将文档划分为区域 (块)。每个块仅仅包含一类数据 (文本、图形、半色调图像等)。每一个都应该只包含一种类型的数据 (文本、图形、半色调图像等)。然后, 计算这些块的一些基本特征。

## 5.7.9 识别验证码

识别验证码首先要把图像二值化，然后识别指定区域的两个数字，最后用和减去其中一个数。例如，识别 <http://www.12306.cn/mormhweb/kyfw/ypcx/> 中的验证码。

//识别数字

```
public static int ocrNumbers() throws Exception {
    String fileName = "D:/search/passCode2_bin.bmp"; //处理二值化后的图片
    File original_f = new File(fileName);
    BufferedImage bi = ImageIO.read(original_f);

    //提取左边的图片对应的数字
    int width = 19;
    int height = bi.getHeight();
    int[] pixels = new int[width * height];
    bi.getRGB(0, 0, width, height, pixels, 0, width);

    BufferedImage leftImg = new BufferedImage(width,
        height, BufferedImage.TYPE_INT_RGB);

    leftImg.setRGB(0, 0, width, height, pixels, 0, width);

    OCR ocr = new OCR();
    ocr.preload();

    String text = ocr.recognize(new Entry(leftImg)); //左边的图片识别出文本
    int leftNum = ChineseConverter.getFigure(text.charAt(0)); //左边的数字

    //提取右边的图片对应的数字
    width = 15;
    height = bi.getHeight();
    pixels = new int[width * height];
    int x = 55;
    bi.getRGB(x, 0, width, height, pixels, 0, width);

    BufferedImage rightImg = new BufferedImage(width,
        height, BufferedImage.TYPE_INT_RGB);

    rightImg.setRGB(0, 0, width, height, pixels, 0, width);
```

```
String rightText = ocr.recognize(new Entry(rightImg));    //右边的图片识别出文本
int sum = Integer.parseInt(rightText); //右边的数字
return (sum-leftNum); //中间的结果
}
```

### 5.7.10 JavaOCR

JavaOCR 是一个纯 Java 实现的图像处理和识别库，下载地址是 <http://sourceforge.net/projects/javaocr/>，可以通过插件来扩展它的功能。

计算所需的图像矩。

定义单个字符对应的簇。使用度量函数来确定点和簇之间的距离。例如，最简单的度量方法可以使用欧几里德度量，JavaOCR 提供马哈拉诺比斯距离，能够得到比欧几里德度量更好的效果。

通过程序自动学习的过程建立起簇。首先收集一些样本，例如，每个字符有 200 个样本。然后开始训练。

- (1) 提取图像特征（在识别过程也是提取同样的特征）。
- (2) 为字符构建初始簇。
- (3) 使用簇分析方法构建在识别阶段实际使用的簇。
- (4) 执行簇匹配来判断解决方案的准确度。
- (5) 调节特征和簇算法直到得到可接受的准确度。
- (6) 得到可接受的配置以后，可以序列化簇对象到 JSON（或者通过其他的方法序列化），然后在实际应用中使用这些对象。每个训练后的字符会有一个或多个簇。
- (7) Hu 利用二阶和三阶中心矩构造了七个不变矩。

## 5.8 提取地域信息

根据 IP 地址或者手机定位用户所在地域。



## 5.8.1 IP 地址

纯真 IP 数据库 (<http://www.cz88.net/>) 可以提供地址和 IP 的对应关系。“纯真 IP 数据库”是一个名为 QQWry.DAT 的文件，它有自己的文件结构。通过了解它的文件结构，可以实现 IP 地址和区域的映射程序。

QQWry.dat 文件在结构上分为三部分：文件头、记录区和索引区。一般我们要查找 IP 时，先在索引区查找记录偏移，之后到记录区读出信息。由于记录区的记录是不定长的，所以直接在记录区中搜索是不可能的。由于记录数比较多，全部遍历索引区也会比较慢，一般来说，我们可以用二分查找法搜索索引区，其速度比遍历索引区快。图 5-14 所示是纯真 IP 数据库的文件结构。

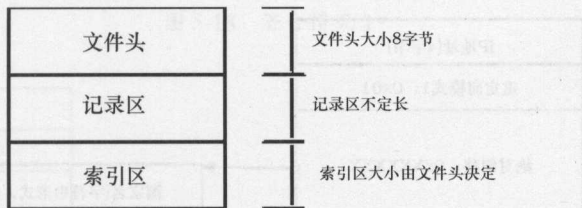


图 5-14 纯真 IP 数据库文件结构

注意，QQWry.dat 文件全部采用了 Little-endian 字节序。QQWry.dat 文件头只有 8 个字节，其结构非常简单，前四个字节是第一条索引的绝对偏移，后四个字节是最后一条索引的绝对偏移。每条 IP 记录都由国家和地区名组成，国家和地区在这里并不是太确切，因为可能会查出“清华大学计算机系”之类的内容，这里清华大学就成了国家名了，因此记录的格式有点像 QName，即由全局部分和局部部分组成，但本书还是沿用国家名和地区名的说法。

在最简单的情况下，一条记录的格式应该是 [IP 地址][国家名][地区名]。国家名和地区名可能会有很多的重复，如果每条记录都保存一个完整的名称复制是非常不理想的，因此就需要重定向以节省空间。为了得到一个国家名或者地区名，就有了两个可能：第一就是直接用字符串表示国家名；第二就是采用一个 4 字节的结构，第一个字节表明了重定向的模式，后面 3 个字节是国家名或者地区名的实际偏移位置。对于国家名来说，情况还可能更复杂些，因为这样的重定向最多可能有两次。

那么什么是重定向模式？根据上面所述，一条记录的格式是 [IP 地址][国家记录][地区记录]，

如果国家记录是重定向，那么地区记录有可能没有，于是就有了两种情况，模式 1 和模式 2。图 5-15 介绍 IP 记录的最简单格式。

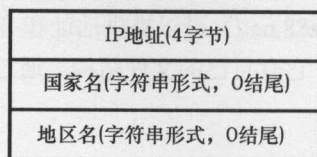


图 5-15 IP 记录的最简单形式

图 5-16 演示了重定向模式 1 的情况。在模式 1 的情况下，地区记录和国家记录一同移走了，后面 3 个字节构成了一个指针，指向了实际的国家名，然后指向地区名。模式 1 的标识字节是 0x01。

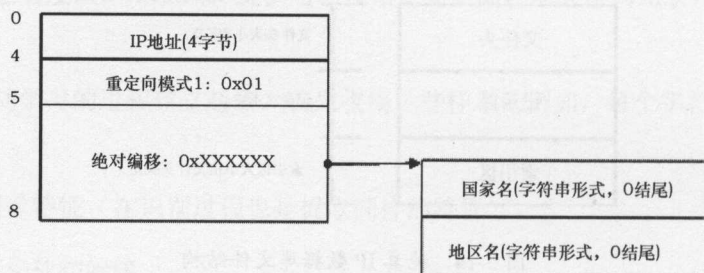


图 5-16 重定向模式 1

图 5-17 演示了重定向模式 2 的情况。在模式 2 的情况下（其标识字节是 0x02），地区记录没有跟着国家记录移走，因此在国家记录之后还有地区记录。模式 1 和模式 2 的区别，模式 1 的国家记录后面不会再有地区记录，模式 2 的国家记录后会有地区记录。

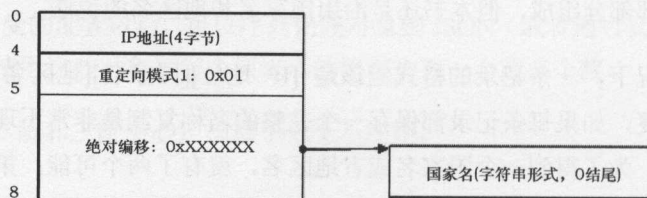


图 5-17 重定向模式 2

图 5-18 演示了当国家记录为模式 1 的时候可能出现的更复杂情况，在这种情况下，重定向指向的位置仍然是个重定向，不过第二次重定向为模式 2。这个重定向最多只有两次，如果发生了第二次重定向，则其一定为模式 2，而且这种情况只会发生在国家记录上。对于地区记录，

模式1和模式2是一样的,地区记录也不会发生两次重定向。不过,图5-18还可以更复杂,变成图5-19和图5-20两种情况。

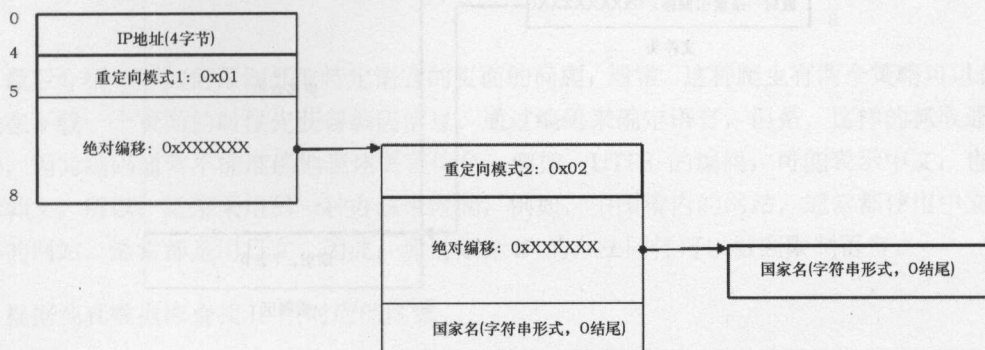


图 5-18 混合情况 1

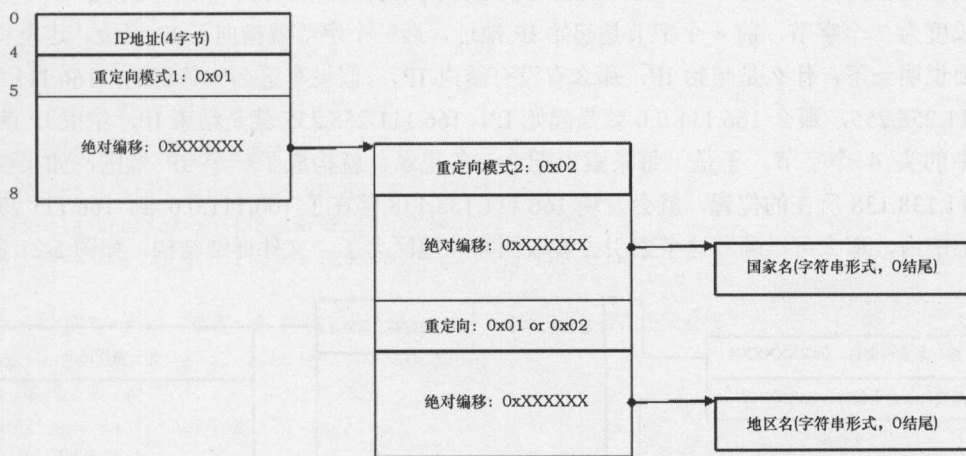


图 5-19 混合情况 2

总之,一条 IP 记录由[IP 地址][国家记录][地区记录]组成。对于国家记录,可以有三种表示方式:字符串形式、重定向模式1和重定向模式2。对于地区记录,可以有两种表示方式:字符串形式和重定向。另外有一条规则:重定向模式1的国家记录后不能跟地区记录。按照这个总结,在这些方式中合理组合,就构成了 IP 记录的所有可能情况。

实际上,文件头是两个指针,分别指向了第一条索引和最后一条索引的绝对偏移,如图5-20所示。



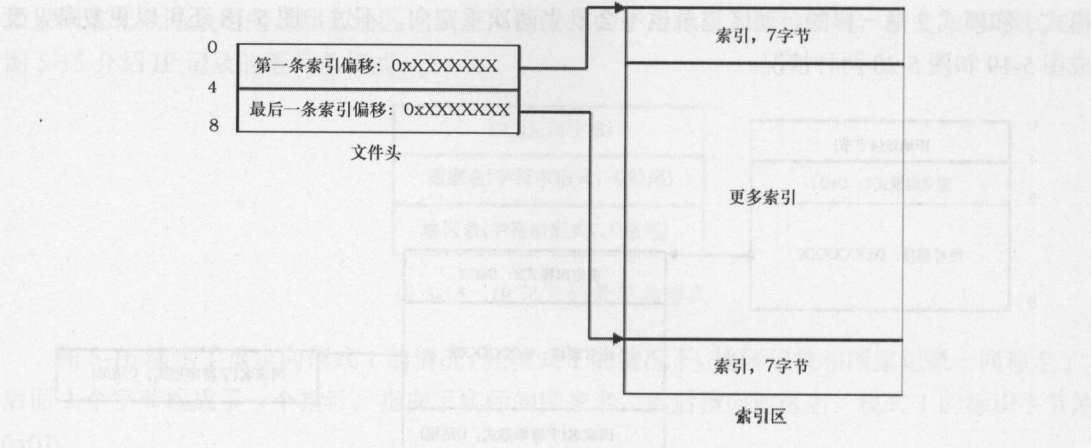


图 5-20 文件头指向索引区图示

实在是很简单，不是吗？从文件头就可以定位到索引区，然后就可以开始搜索 IP 了。每条索引长度为 7 个字节，前 4 个字节是起始 IP 地址，后三个字节就指向了 IP 记录。这里有些概念需要说明一下，什么是起始 IP，那么有没有结束 IP？假设有这么一条记录：166.111.0.0 到 166.111.255.255，那么 166.111.0.0 就是起始 IP，166.111.255.255 就是结束 IP，结束 IP 就是 IP 记录中的头 4 个字节。于是，每条索引配合一条记录，就构成了一个 IP 范围，如果要查找 166.111.138.138 所在的位置，就会发现 166.111.138.138 落在了 166.111.0.0 到 166.111.255.255 这个范围内，那么可以顺着这条索引去读取国家和地区名了。文件详细结构，如图 5-21 所示。

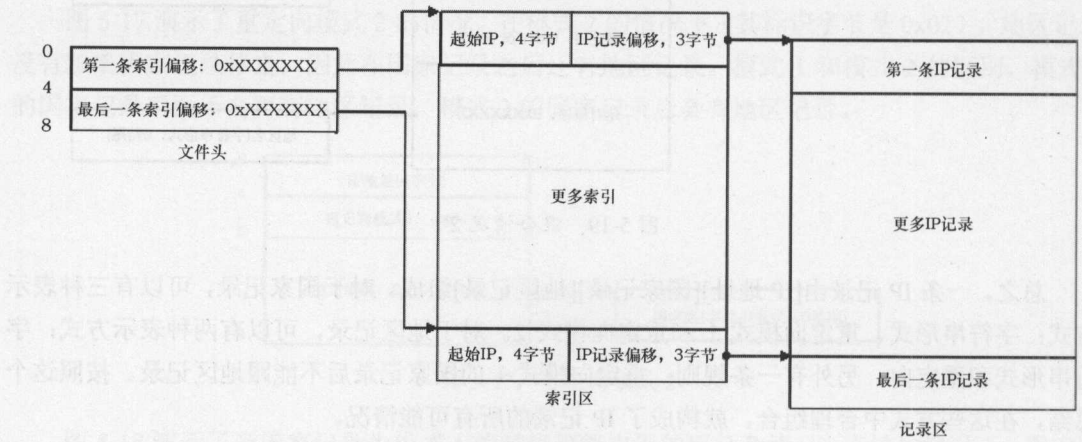


图 5-21 文件详细结构

QQWry.dat 的版本信息保存在哪里了呢？答案是：最后一条 IP 记录实际上就是版本信息，最后一条记录显示的样子是：255.255.255.0 255.255.255.255 纯真网络 2004 年 6 月 25 日 IP 数据。

最后介绍一下如何限制抓取特定语言的页面的问题，通常，这种爬虫有两个策略可以使用。一是在下载一个页面的时候先获得编码信息，通过编码来确定语言，但是，这样的抓取通常不准确，因为编码通常不能准确地表述语言信息，例如，UTF8 的编码，可能表示中文，也可能表示韩文。所以，通常采用另一种办法来限制，例如，中国境内的网站，通常都使用中文。而日本的网站，通常都是用日文。因此，通过限制 IP 的办法同样可以做到限制语言。

根据纯真数据库查找 IP 所对应的区域。

```
public class IPSeeker {
    //纯真 IP 数据库名
    private String IP_FILE="QQWry.Dat";
    //保存的文件夹
    private String INSTALL_DIR="c:\\qqwry";

    //一些固定常量，如记录长度等
    private static final int IP_RECORD_LENGTH = 7;
    private static final byte REDIRECT_MODE_1 = 0x01;
    private static final byte REDIRECT_MODE_2 = 0x02;

    //用来做为 cache，查询一个 IP 时首先查看 cache，以减少不必要的重复查找
    private Map<String, IPLocation> ipCache;
    //随机文件访问类
    private RandomAccessFile ipFile;
    //内存映射文件
    private MappedByteBuffer mbb;
    //起始地区的开始和结束的绝对偏移
    private long ipBegin, ipEnd;
    //为高效率而采用的临时变量
    private IPLocation loc;
    private byte[] buf;
    private byte[] b4;
    private byte[] b3;

    public IPSeeker(String fileName,String dir) {
        this.INSTALL_DIR=dir;
```

```

this.IP_FILE=fileName;
ipCache = new HashMap<String, IPLocation>();
loc = new IPLocation();
buf = new byte[100];
b4 = new byte[4];
b3 = new byte[3];
try {
    ipFile = new RandomAccessFile(IP_FILE, "r");
} catch (FileNotFoundException e) {
    //如果找不到这个文件，再尝试在当前目录下搜索，这次全部改用小写文件名
    //因为有些系统可能会区分大小写，导致找不到IP地址信息文件
    String filename = new File(IP_FILE).getName().toLowerCase();
    File[] files = new File(INSTALL_DIR).listFiles();
    for(int i = 0; i < files.length; i++) {
        if(files[i].isFile()) {
            if(files[i].getName().toLowerCase().equals(filename)) {
                try {
                    ipFile = new RandomAccessFile(files[i], "r");
                } catch (FileNotFoundException e1) {
                    LogFactory.log("IP地址信息文件没有找到，IP显示功能
                        将无法使用", Level.ERROR, e1);
                    ipFile = null;
                }
                break;
            }
        }
    }
}
//如果打开文件成功，读取文件头信息
if(ipFile != null) {
    try {
        ipBegin = readLong4(0);
        ipEnd = readLong4(4);
        if(ipBegin == -1 || ipEnd == -1) {
            ipFile.close();
            ipFile = null;
        }
    } catch (IOException e) {
        LogFactory.log("IP地址信息文件格式有错误，IP显示功能将无法使用",
            Level.ERROR, e);
        ipFile = null;
    }
}

```



```

    }
}

/**
 * 给定一个地点的不完全名字，得到一系列包含 s 子串的 IP 范围记录
 * @param s 地点子串
 * @return 包含 IPEntry 类型的 List
 */
public List getIPEntriesDebug(String s) {
    List<IPEntry> ret = new ArrayList<IPEntry>();
    long endOffset = ipEnd + 4;
    for(long offset = ipBegin + 4; offset <= endOffset; offset +=
    IP_RECORD_LENGTH) {
        //读取结束 IP 偏移
        long temp = readLong3(offset);
        //如果 temp 不等于-1，读取 IP 的地点信息
        if(temp != -1) {
            IPLocation ipLoc = getIPLocation(temp);
            //判断这个地点里面是否包含了 s 子串，如果包含，添加这个记录到 List 中
            //如果没有，继续
            if(ipLoc.getCountry().indexOf(s) != -1 ||
            ipLoc.getArea().indexOf(s) != -1) {
                IPEntry entry = new IPEntry();
                entry.country = ipLoc.getCountry();
                entry.area = ipLoc.getArea();
                //得到起始 IP
                readIP(offset - 4, b4);
                entry.beginIp = Util.getIpStringFromBytes(b4);
                //得到结束 IP
                readIP(temp, b4);
                entry.endIp = Util.getIpStringFromBytes(b4);
                //添加该记录
                ret.add(entry);
            }
        }
    }
    return ret;
}

```

```

public IPLocation getLocation(String ip){
    IPLocation location=new IPLocation();
    location.setArea(this.getArea(ip));
    location.setCountry(this.getCountry(ip));
    return location;
}

/**
 * 给定一个地点的不完全名字，得到一系列包含 s 子串的 IP 范围记录
 * @param s 地点子串
 * @return 包含 IPEntry 类型的 List
 */
public List<IPEntry> getIPEntries(String s) {
    List<IPEntry> ret = new ArrayList<IPEntry>();
    try {
        //映射 IP 信息文件到内存中
        if(mbb == null) {
            FileChannel fc = ipFile.getChannel();
            mbb = fc.map(FileChannel.MapMode.READ_ONLY, 0,
                ipFile.length());
            mbb.order(ByteOrder.LITTLE_ENDIAN);
        }

        int endOffset = (int)ipEnd;
        for(int offset = (int)ipBegin + 4; offset <= endOffset; offset +=
            IP_RECORD_LENGTH) {
            int temp = readInt3(offset);
            if(temp != -1) {
                IPLocation ipLoc = getLocation(temp);
                //判断这个地点里面是否包含 s 子串，如果包含，在 List 中添加这个记录
                //如果没有，继续
                if(ipLoc.getCountry().indexOf(s) != -1 ||
                    ipLoc.getArea().indexOf(s) != -1) {
                    IPEntry entry = new IPEntry();
                    entry.country = ipLoc.getCountry();
                    entry.area = ipLoc.getArea();
                    //得到起始 IP
                    readIP(offset - 4, b4);
                    entry.beginIp = Util.getIpStringFromBytes(b4);
                    //得到结束 IP
                    readIP(temp, b4);
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return ret;
}

```

```

        entry.endIp = Util.getIpStringFromBytes(b4);
        //添加该记录
        ret.add(entry);
    }
}

} catch (IOException e) {
    LogFactory.log("", Level.ERROR, e);
}
return ret;
}

/**
 * 从内存映射文件 offset 位置开始的第 3 个字节起读取一个 int
 * @param offset
 * @return
 */
private int readInt3(int offset) {
    mbb.position(offset);
    return mbb.getInt() & 0x00FFFFFF;
}

/**
 * 从内存映射文件当前位置开始的第 3 个字节起读取一个 int
 * @return
 */
private int readInt3() {
    return mbb.getInt() & 0x00FFFFFF;
}

/**
 * 根据 IP 得到国家名
 * @param ip IP 的字节数组形式
 * @return 国家名字符串
 */
public String getCountry(byte[] ip) {
    //检查 IP 地址文件是否正常
    if(ipFile == null)
        return Message.bad_ip_file;
    //保存 IP, 转换 IP 字节数组为字符串形式
    String ipStr = Util.getIpStringFromBytes(ip);

```



```

        //先检查 cache 中是否已经包含这个 IP 的结果, 没有再搜索文件
        if (ipCache.containsKey(ipStr)) {
            IPLocation ipLoc = ipCache.get(ipStr);
            return ipLoc.getCountry();
        } else {
            IPLocation ipLoc = getIPLocation(ip);
            ipCache.put(ipStr, ipLoc.getCopy());
            return ipLoc.getCountry();
        }
    }

    /**
     * 根据 IP 得到国家名
     * @param ip IP 的字符串形式
     * @return 国家名字符串
     */
    public String getCountry(String ip) {
        return getCountry(Util.getIpByteArrayFromString(ip));
    }

    /**
     * 根据 IP 得到地区名
     * @param ip IP 的字节数组形式
     * @return 地区名字符串
     */
    public String getArea(byte[] ip) {
        //检查 IP 地址文件是否正常
        if (ipFile == null)
            return Message.bad_ip_file;
        //保存 IP, 转换 IP 字节数组为字符串形式
        String ipStr = Util.getIpStringFromBytes(ip);
        //先检查 cache 中是否已经包含这个 IP 的结果, 没有再搜索文件
        if (ipCache.containsKey(ipStr)) {
            IPLocation ipLoc = ipCache.get(ipStr);
            return ipLoc.getArea();
        } else {
            IPLocation ipLoc = getIPLocation(ip);
            ipCache.put(ipStr, ipLoc.getCopy());
            return ipLoc.getArea();
        }
    }
}

```

```

/**
 * 根据 IP 得到地区名
 * @param ip IP 的字符串形式
 * @return 地区名字符串
 */
public String getArea(String ip) {
    return getArea(Util.getIpByteArrayFromString(ip));
}

/**
 * 根据 ip 搜索 IP 信息文件, 得到 IPLocation 结构, 所搜索的 ip 参数从类成员 ip 中得到
 * @param ip 要查询的 IP
 * @return IPLocation 结构
 */
private IPLocation getIPLocation(byte[] ip) {
    IPLocation info = null;
    long offset = locateIP(ip);
    if(offset != -1)
        info = getIPLocation(offset);
    if(info == null) {
        info = new IPLocation();
        info.setCountry (Message.unknown_country);
        info.setArea (Message.unknown_area);
    }
    return info;
}

/**
 * @param offset
 * @return 读取的 long 值, 返回-1 表示读取文件失败
 */
private long readLong4(long offset) {
    long ret = 0;
    try {
        ipFile.seek(offset);
        ret |= (ipFile.readByte() & 0xFF);
        ret |= ((ipFile.readByte() << 8) & 0xFF00);
        ret |= ((ipFile.readByte() << 16) & 0xFF0000);
        ret |= ((ipFile.readByte() << 24) & 0xFF000000);
        return ret;
    }
}

```



```

        } catch (IOException e) {
            return -1;
        }
    }

    /**
     * @param offset 整数的起始偏移
     * @return 读取的 long 值, 返回-1 表示读取文件失败
     */
    private long readLong3(long offset) {
        long ret = 0;
        try {
            ipFile.seek(offset);
            ipFile.readFully(b3);
            ret |= (b3[0] & 0xFF);
            ret |= ((b3[1] << 8) & 0xFF00);
            ret |= ((b3[2] << 16) & 0xFF0000);
            return ret;
        } catch (IOException e) {
            return -1;
        }
    }

    /**
     * 从当前位置读取 3 个字节转换成 long
     * @return 读取的 long 值, 返回-1 表示读取文件失败
     */
    private long readLong3() {
        long ret = 0;
        try {
            ipFile.readFully(b3);
            ret |= (b3[0] & 0xFF);
            ret |= ((b3[1] << 8) & 0xFF00);
            ret |= ((b3[2] << 16) & 0xFF0000);
            return ret;
        } catch (IOException e) {
            return -1;
        }
    }
}

/**

```



\* 从 offset 位置读取 4 个字节的 IP 地址放入 ip 数组中, 读取后的 ip 为 Big-endian 格式, 但是文件中是 Little-endian 形式, 将会进行转换

\* @param offset

\* @param ip

\*/

```
private void readIP(long offset, byte[] ip) {
```

```
    try {
```

```
        ipFile.seek(offset);
```

```
        ipFile.readFully(ip);
```

```
        byte temp = ip[0];
```

```
        ip[0] = ip[3];
```

```
        ip[3] = temp;
```

```
        temp = ip[1];
```

```
        ip[1] = ip[2];
```

```
        ip[2] = temp;
```

```
    } catch (IOException e) {
```

```
        LogFactory.log("", Level.ERROR, e);
```

```
    }
```

```
}
```

/\*\*

\* 从 offset 位置读取 4 个字节的 IP 地址放入 ip 数组中, 读取后的 ip 为 Big-endian 格式

\* 式, 但是文件中是 little-endian 形式, 将会进行转换

\* @param offset

\* @param ip

\*/

```
private void readIP(int offset, byte[] ip) {
```

```
    mbb.position(offset);
```

```
    mbb.get(ip);
```

```
    byte temp = ip[0];
```

```
    ip[0] = ip[3];
```

```
    ip[3] = temp;
```

```
    temp = ip[1];
```

```
    ip[1] = ip[2];
```

```
    ip[2] = temp;
```

```
}
```

/\*\*

\* 把类成员 ip 和 beginIp 比较, 注意这个 beginIp 是 Big-endian 格式的

\* @param ip 要查询的 IP

\* @param beginIp 和被查询 IP 相比较的 IP

```

    * @return 若 ip 和 begin 相等返回 0, ip 大于 beginIp 则返回 1, 小于返回-1
    */
private int compareIP(byte[] ip, byte[] beginIp) {
    for(int i = 0; i < 4; i++) {
        int r = compareByte(ip[i], beginIp[i]);
        if(r != 0)
            return r;
    }
    return 0;
}

/**
 * 把两个 byte 当作无符号数进行比较
 * @param b1
 * @param b2
 * @return 若 b1 大于 b2 则返回 1, 相等返回 0, 小于返回-1
 */
private int compareByte(byte b1, byte b2) {
    if((b1 & 0xFF) > (b2 & 0xFF)) //比较是否大于
        return 1;
    else if((b1 ^ b2) == 0) //判断是否相等
        return 0;
    else
        return -1;
}

/**
 * 这个方法将根据 IP 的内容, 定位到包含这个 IP 国家地区的记录处, 返回一个绝对偏移
 * 使用二分法查找
 * @param ip 要查询的 IP
 * @return 如果找到了, 返回结束 IP 的偏移, 如果没有找到, 返回-1
 */
private long locateIP(byte[] ip) {
    long m = 0;
    int r;
    //比较第一个 ip 项
    readIP(ipBegin, b4);
    r = compareIP(ip, b4);
    if(r == 0) return ipBegin;
    else if(r < 0) return -1;
    //开始二分搜索

```

```

for(long i = ipBegin, j = ipEnd; i < j; ) {
    m = getMiddleOffset(i, j);
    readIP(m, b4);
    r = compareIP(ip, b4);
    // log.debug(Utils.getIpStringFromBytes(b));
    if(r > 0)
        i = m;
    else if(r < 0) {
        if(m == j) {
            j -= IP_RECORD_LENGTH;
            m = j;
        } else
            j = m;
    } else
        return readLong3(m + 4);
}
//如果循环结束了, 那么 i 和 j 必定是相等的, 这个偏移量对应的记录为最可能的记录
//但是并非肯定就是, 所以还要检查一下这个记录, 如果是, 就返回结束地址区的绝对偏移
m = readLong3(m + 4);
readIP(m, b4);
r = compareIP(ip, b4);
if(r <= 0) return m;
else return -1;
}

/**
 * 得到 begin 偏移和 end 偏移中间位置记录的偏移
 * @param begin
 * @param end
 * @return
 */
private long getMiddleOffset(long begin, long end) {
    long records = (end - begin) / IP_RECORD_LENGTH;
    records >>= 1;
    if(records == 0) records = 1;
    return begin + records * IP_RECORD_LENGTH;
}

/**
 * 给定一个 IP 国家地区记录的偏移, 返回一个 IPLocation 结构
 * @param offset 国家记录的起始偏移

```



```

    * @return IPLocation 对象
    */
private IPLocation getIPLocation(long offset) {
    try {
        //跳过 4 字节 ip
        ipFile.seek(offset + 4);
        //读取第一个字节判断是否标志字节
        byte b = ipFile.readByte();
        if(b == REDIRECT_MODE_1) {
            //读取国家偏移
            long countryOffset = readLong3();
            //跳转至偏移处
            ipFile.seek(countryOffset);
            //再检查一次标志字节，因为这个时候这个地方仍然可能是个重定向
            b = ipFile.readByte();
            if(b == REDIRECT_MODE_2) {
                loc.setCountry ( readString(readLong3()));
                ipFile.seek(countryOffset + 4);
            } else
                loc.setCountry ( readString(countryOffset));
            //读取地区标志
            loc.setArea( readArea(ipFile.getFilePointer()));
        } else if(b == REDIRECT_MODE_2) {
            loc.setCountry ( readString(readLong3()));
            loc.setArea( readArea(offset + 8));
        } else {
            loc.setCountry ( readString(ipFile.getFilePointer() - 1));
            loc.setArea( readArea(ipFile.getFilePointer()));
        }
        return loc;
    } catch (IOException e) {
        return null;
    }
}

/**
 * 给定一个 IP 国家地区记录的偏移，返回一个 IPLocation 结构，此方法应用于内存映射文件方式
 * @param offset 国家记录的起始偏移
 * @return IPLocation 对象
 */
private IPLocation getIPLocation(int offset) {

```

```

//跳过4字节IP
mbb.position(offset + 4);
//判断第一个字节是否为标志字节
byte b = mbb.get();
if(b == REDIRECT_MODE_1) {
    //读取国家偏移
    int countryOffset = readInt3();
    //跳转至偏移处
    mbb.position(countryOffset);
    //再检查一次标志字节，因为这个时候这个地方仍然可能是个重定向
    b = mbb.get();
    if(b == REDIRECT_MODE_2) {
        loc.setCountry ( readString(readInt3()));
        mbb.position(countryOffset + 4);
    } else
        loc.setCountry(readString(countryOffset));
    //读取地区标志
    loc.setArea(readArea(mbb.position()));
} else if(b == REDIRECT_MODE_2) {
    loc.setCountry(readString(readInt3()));
    loc.setArea(readArea(offset + 8));
} else {
    loc.setCountry(readString(mbb.position() - 1));
    loc.setArea(readArea(mbb.position()));
}
return loc;
}

/**
 * 从offset 偏移开始解析后面的字节，读出一个地区名
 * @param offset 地区记录的起始偏移
 * @return 地区名字符串
 * @throws IOException
 */
private String readArea(long offset) throws IOException {
    ipFile.seek(offset);
    byte b = ipFile.readByte();
    if(b == REDIRECT_MODE_1 || b == REDIRECT_MODE_2) {
        long areaOffset = readLong3(offset + 1);
        if(areaOffset == 0)
            return Message.unknown_area;
    }
}

```

```

        else
            return readString(areaOffset);
    } else
        return readString(offset);
}

/**
 * @param offset 地区记录的起始偏移
 * @return 地区名字符串
 */
private String readArea(int offset) {
    mbb.position(offset);
    byte b = mbb.get();
    if(b == REDIRECT_MODE_1 || b == REDIRECT_MODE_2) {
        int areaOffset = readInt3();
        if(areaOffset == 0)
            return Message.unknown_area;
        else
            return readString(areaOffset);
    } else
        return readString(offset);
}

/**
 * 从 offset 偏移处读取一个以 0 结束的字符串
 * @param offset 字符串起始偏移
 * @return 读取的字符串, 出错返回空字符串
 */
private String readString(long offset) {
    try {
        ipFile.seek(offset);
        int i;
        for(i = 0, buf[i] = ipFile.readByte(); buf[i] != 0; buf[++i] =
            ipFile.readByte());
        if(i != 0)
            return Util.getString(buf, 0, i, "GBK");
    } catch (IOException e) {
        LogFactory.log("", Level.ERROR, e);
    }
    return "";
}
}

```



```

/**
 * 从内存映射文件的 offset 位置得到一个0 结尾字符串
 * @param offset 字符串起始偏移
 * @return 读取的字符串, 出错返回空字符串
 */
private String readString(int offset) {
    try {
        mbb.position(offset);
        int i;
        for(i = 0, buf[i] = mbb.get(); buf[i] != 0; buf[++i] = mbb.get());
        if(i != 0)
            return Util.getString(buf, 0, i, "GBK");
    } catch (IllegalArgumentException e) {
        LogFactory.log("", Level.ERROR, e);
    }
    return "";
}

}

/**
 * 工具类, 提供一些方便的方法
 */
public class Util {

    private static StringBuilder sb = new StringBuilder();

    /**
     * 从IP 的字符串形式得到字节数组形式
     * @param ip 字符串形式的 IP
     * @return 字节数组形式的 IP
     */
    public static byte[] getIpByteArrayFromString(String ip) {
        byte[] ret = new byte[4];
        StringTokenizer st = new StringTokenizer(ip, ".");
        try {
            ret[0] = (byte) (Integer.parseInt(st.nextToken()) & 0xFF);
            ret[1] = (byte) (Integer.parseInt(st.nextToken()) & 0xFF);
            ret[2] = (byte) (Integer.parseInt(st.nextToken()) & 0xFF);
            ret[3] = (byte) (Integer.parseInt(st.nextToken()) & 0xFF);
        } catch (Exception e) {

```

```

        LogFactory.log("从 IP 的字符串形式得到字节数组形式报错", Level.ERROR, e);
    }
    return ret;
}

/**
 * @param ip
 * IP 的字节数组形式
 * @return 字符串形式的 IP
 */
public static String getIpStringFromBytes(byte[] ip) {
    sb.delete(0, sb.length());
    sb.append(ip[0] & 0xFF);
    sb.append('.');
    sb.append(ip[1] & 0xFF);
    sb.append('.');
    sb.append(ip[2] & 0xFF);
    sb.append('.');
    sb.append(ip[3] & 0xFF);
    return sb.toString();
}

/**
 * 根据某种编码方式将字节数组转换成字符串
 * @param b 字节数组
 * @param offset 要转换的起始位置
 * @param len 要转换的长度
 * @param encoding 编码方式
 * @return 如果 encoding 不支持, 返回一个默认编码的字符串
 */
public static String getString(byte[] b, int offset, int len,
    String encoding) {
    try {
        return new String(b, offset, len, encoding);
    } catch (UnsupportedEncodingException e) {
        return new String(b, offset, len);
    }
}

}

public interface Message {

```

```

String bad_ip_file="IP 地址库文件错误";
String unknown_country="未知国家";
String unknown_area="未知地区";
}

/**
 * <pre>
 * 一条 IP 范围记录，不仅包括国家和区域，也包括起始 IP 和结束 IP
 * </pre>
 */
public class IPEntry {
    public String beginIp;
    public String endIp;
    public String country;
    public String area;

    /**
     * 构造函数
     */
    public IPEntry() {
        beginIp = endIp = country = area = "";
    }
}

/**
 * @category 用来封装 IP 相关信息，目前只有两个字段，IP 所在的国家和地区
 */

public class IPLocation {
    private String country;
    private String area;

    public IPLocation() {
        country = area = "";
    }

    public IPLocation getCopy() {
        IPLocation ret = new IPLocation();
        ret.country = country;
        ret.area = area;
        return ret;
    }
}

```



```
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public String getArea() {
        return area;
    }

    public void setArea(String area) {
        //如果为局域网, 纯真 IP 地址库的地区会显示 CZ88.NET, 这里把它去掉
        if(area.trim().equals("CZ88.NET")){
            this.area="本机或本网络";
        }else{
            this.area = area;
        }
    }
}

/**
 * 日志工厂
 */
public class LogFactory {
    private static final Logger logger;
    static {
        logger = Logger.getLogger("stdout");
        logger.setLevel(Level.DEBUG);
    }

    public static void log(String info, Level level, Throwable ex) {
        logger.log(level, info, ex);
    }

    public static Level getLogLevel() {
        return logger.getLevel();
    }
}
```

```

}

public class IPTest {

    public static void main(String[] args) {
        //指定纯真数据库的文件名和所在文件夹
        IPSeeker ip = new IPSeeker("QQWry.Dat", "c:\\qqwry");
        //测试 IP 58.20.43.13
        System.out.println(ip.getIPLocation("58.20.43.13").getCountry() + ":"
            + ip.getIPLocation("58.20.43.13").getArea());
    }
}

```

运行程序，输出如下所示。

湖南省长沙市:联通

说明 IP 地址为 58.20.43.13 的区域属于湖南省长沙市。如果想限制不抓取湖南省长沙市的 IP，那就不能抓取这个主机的文件。

## 5.8.2 手机

出去玩的时候需要知道出发地和目的地。出发地组织人往往会提供手机号码，从手机号码可以提取地域信息，根据手机号码的前 7 位判断所属地域。取得地域信息的另外一个方法是根据用户 IP 地址判断。

```

private static final String QUERY_SQL = "select city from mobileloc where num = ?";
//根据手机号得到所属城市
public static String getStartCityByPhone(String mobile) {
    String startCity = null;
    if(mobile.trim().length() == 11) {
        mobile = mobile.substring(0,7);//根据前 7 位判断
        ResultSet rs = DBManager.getInstance().
            executeQueryByAdd(QUERY_SQL, mobile);
        if(rs.next()) {
            startCity = rs.getString("city");
        }
    }
    return startCity;
}

```

//测试方法

```
public static void main(String[] args) {  
    System.out.println(getStartCityByPhone("18903045009"));  
}
```

## 5.9 提取新闻

从新闻中提取“时间、地点、人物、事件、后效”等。例如，英国《每日邮报》的网站报道。

科学家近日发出警告，表示 2030 年，太阳将进入‘休眠期’，这会让地球步入‘迷你冰河期’。据瓦伦蒂娜·扎尔科夫教授及其研究团队在英国皇家天文学会于威尔士兰迪德诺召开的国家天文会议上公布的研究成果，太阳活动在 2030 年左右会减少 60%。这让人想起以地球遭受极寒肆虐的天气为背景的电影《雪国列车》。如果地球真的陷入极寒，对人类社会的影响必定是天翻地覆的。

在远古时代，地球曾一度进入冰川时期。有一种说法认为，在约 195 000 年前开始的冰川时期，远古智人才刚刚出现，剧烈的气候变化导致的恶劣环境几乎毁灭了地球上的所有人，只有极少数人类幸存下来，而在冰川灾难中存活下来的人，成了我们的祖先。

如今科技有了长足的发展，但是如果冰河时期再次降临，人类能够安然度过吗？

- 时间：2030 年。
- 地点：太阳。
- 事件：太阳将进入“休眠期”。
- 后效：地球步入“迷你冰河期”。

首先分句，然后确定句子间的关系。从陈述句中提取时间、地点、人物、事件等的关键要素，根据句子间关系提取后效句子。

提取时间的代码如下所示。

```
ExtractGrammar g = new ExtractGrammar();  
  
String right = "<date>{time}";  
g.add("时间", right);
```



```
TextExtractor ie = new TextExtractor(g);
```

```
String news = "英国《每日邮报》的网站报道，科学家近日发出警告，表示2030年，太阳将进入“休眠期”，这会  
让地球步入“迷你冰河期”。据瓦伦蒂娜·扎尔科夫教授及其研究团队在英国皇家天文学会于威尔士兰迪德诺召开的国家  
天文会议上公布的研究成果，太阳活动在2030年左右会减少60%。这让人想起以地球遭受极寒肆虐天气为背景的电影《雪  
国列车》。如果地球真的陷入极寒，对人类社会的影响必定是天翻地覆的。";
```

```
AdjList adjList = ie.getLattice(news);
```

```
String year = adjList.args.getFirst("time");
```

```
System.out.println(year); //输出时间：2030年
```

## 5.10 流媒体内容提取

相对于文本检索，对音频和视频检索技术研究得比较少。常用的方法是提取出相关的文字描述来索引音频和视频。例如，把视频里面的声音通过语音识别（Speech Recognition），然后放入索引库，同时记录时间点。语音识别的方法有很多技术问题，比如电影里面有背景音乐，去除噪声和音乐比较麻烦，而且多语言混杂，需要多种语言处理包，说话的人可能有各种方言，这些都导致了转换率非常低。此外，把 rmvb 等视频文件格式语言中的字用屏幕文字识别转换也非常麻烦，因为它涉及多语言的转换，尤其是汉字这样的大字符集文字，除非写得很标准。

视频搜索可以用在电台或电视台制作节目上，他们往往有录制了几十年的数据，现在想从中找出一些内容做合集，语音搜索和人脸搜索可以用上；视频网站每天都会有很多用户上传内容，其中可能有一些违法的内容，如果全部人工审核，成本很大。先用内容检索过滤一遍可以大大降低人工参与的工作量；用户个人拍录像时可以加一些语音标签，方便日后编辑。

face.com 的 API 接口实现的人脸识别，参见 <http://developers.face.com/account/>。

### 5.10.1 音频流内容提取

音频是多媒体中的一种重要媒体。我们能够听见的音频频率范围是 60Hz~20kHz，其中语音大约分布在 300Hz~4kHz 之内，而音乐和其他自然声响是全范围分布的。男人说话声音低沉，因为声带振动频率较低。而女人说话声音尖细，因为声带振动频率较高。童声高音频率范围为

260Hz~880Hz, 低音频率范围 196Hz~700Hz。女声高音频率范围为 220Hz~1.1KHz, 低音频率范围为 200Hz~700Hz。男声高音频率范围为 160Hz~523Hz, 低音频率范围为 80Hz~358Hz。声音的响度对应强弱, 而音高对应频率。频率是声音的物理特性, 而音调则是频率的主观反映。

声音经过模拟设备记录或再生成为模拟音频, 再经数字化成为数字音频。数字化时的采样率必须高于信号带宽的 2 倍, 才能正确恢复信号。样本可用 8 位或 16 位比特表示, 一般保存成 wav 文件格式。尽管 wav 文件可以包括压缩的声音信号, 但是一般情况下是没有压缩的。

以前的许多研究工作涉及语音信号的处理, 如语音识别。当前容易自动识别孤立的字词, 例如用在专用的听写和电话应用方面, 而对连续的语音识别则较困难, 错误较多, 但目前在这方面已经取得了突破性进展。同时还有些研究辨别说话人的技术。

语音检索是以语音为中心的检索, 采用语音识别等处理技术, 可以检索如电台节目、电话交谈、会议录音等。许多语音信号处理的研究成果可以用于语音检索。

#### (1) 利用大词汇语音识别技术进行检索。

这种方法是利用语音识别技术把语音转换为文本, 从而可以采用文本检索方法进行检索。虽然好的连续语音识别系统在小心地操作下可以达到 90% 以上的词语正确度, 但在实际应用中, 如电话和新闻广播等, 识别率并不高。即使这样, 自动语音识别出来的文本仍然对信息检索有用, 这是因为检索任务只是匹配包含在音频数据中的查询词句, 而不是要求一篇可读性好的文章。例如, 采用这种方法把视频的语音对话轨迹转换为文本脚本, 然后组织成适合全文检索的形式支持检索。

#### (2) 基于识别关键词进行检索。

在无约束的语音中自动检测词或短语通常称为关键词的发现 (Spotting)。利用该技术, 识别或标记出长段录音或音轨中反映用户感兴趣的事件, 这些标记就可以用于检索。如通过捕捉体育比赛解说词中词语“进球”可以标记进球的内容。

#### (3) 基于说话人的辨认进行分割。

这种技术是简单地辨别出说话人语音的差别, 而不是识别出说的是什么, 叫作声纹识别。它在合适的环境中可以做到非常准确, 将来也许可以用声纹识别代替刷卡来识别人。利用这种技术, 可以根据说话人的变化分割录音, 并建立录音索引。用这种技术检测视频或多媒体资源的声音轨迹中的说话人的变化, 建立索引和确定某种类型的结构 (如对话)。例如, 分割和分析会议录音, 分割的区段对应于不同的说话人, 可以方便地浏览长篇的会议资料。

Sphinx-4 (<http://cmusphinx.sourceforge.net/>) 是采用 Java 实现的一个语音识别软件。Sphinx 是一个基于隐马尔科夫模型的系统，首先它需要学习一套语音单元的特征，然后根据所学来推断出所需要识别的语音信号最可能的结果，这在学习语音单元特征的过程叫作训练。应用所学来识别语音的过程有时也被称为解码。在 Sphinx 系统中，训练部分由 Sphinx Trainer 来完成，解码部分由 Sphinx Decoder 来完成。为了识别普通话，可以使用 Sphinx Trainer 建立普通话的声学模型。训练时需要准备好语音信号 (Acoustic Signals)，及与训练用语音信号对应的文本 (Transcript File)。当前 Sphinx-4 只能使用 Sphinx-3 Trainer 生成的 Sphinx-3 声学模型。有计划创建 Sphinx-4 trainer 用来生成 Sphinx-4 专门的声学模型，但是这个工作还没完成。

讲稿 (Transcript) 文件中记录了单词和非讲话声的序列。序列接着一个标记可以把这个序列和对应的语音信号关联起来。

例如，有 160 个 wav 文件，每个文件对应一个句子的发音。播放第一个声音文件，会听到 “a player threw the ball to me”，而且就这一句话，可以把这些 wav 或者 raw 格式的声音文件放到 myasm/wav 目录中。

其次，需要一个控制文件。控制文件只是一个文本文件，把控制文件命名为 myam\_train.fileids (必须把它命名成 [name]\_train.fileids 的形式，[name] 是任务的名字，例如，myam)。各声音文件的名字如下所示 (注意，没有文件扩展名)。

```
0001
0002
0003
0004
```

再次，需要一个讲稿文件，文件中的每行有一个独立文件的发声。必须和控制文件相对应。例如，如果控制文件中第一行是 0001，那么讲稿文件中的第一行的讲稿就是 “A player threw the ball to me”，因为这是 0001.wav 的讲稿。讲稿文件也是一个文本文件，命名成 myam.corpus，应该有和控制文件同样的行数。讲稿不包括标点符号，所以要删除任何标题符号，如下所示。

```
a player threw the ball to me
does he like to swim out to sea
how many fish are in the water
you are a good kind of person
```

讲稿文件的顺序对应 0001、0002、0003 和 0004 文件。

现在有了一些声音文件、一个控制文件和一个讲稿文件。



Sphinx-4 由 3 个主要模块组成：前端处理器（FrontEnd）、解码器（Decoder）、语言处理器（Linguist）。前端把一个或多个输入信号参数化成特征序列。语言处理器把任何类型的标准语言模型和声学模型以及词典中的发声信息转换为搜索图。这里，声学模型用来表示字符如何发音，语言模型用来评估一个句子的概率。解码器中的搜索管理器使用前端处理器生成的特征执行实际的解码，生成结果。在识别之前或识别过程中，应用程序都可以发出对每个模块的控制，这样就可以有效的参与到识别过程中来，如图 5-22 所示。

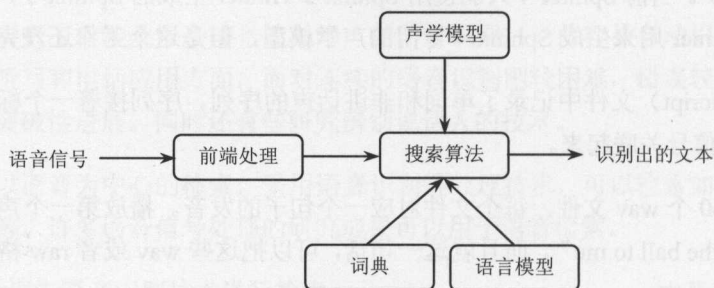


图 5-22 Sphinx-4 结构

语音识别的准确率受限于其识别的内容，内容越简单，则识别准确率越高，所以一般根据某个应用场景来识别语音。例如，电视台给录制的新闻节目加字幕，有批处理和实时翻译两种方式，这里采用批处理的方式。以识别新闻节目为例，开发流程如下所示。

(1) 准备新闻语料库：语料库就是一个文本文件，每行一个句子。

(2) 创建语言模型：一般采用基于统计的  $n$  元语言模型，例如，ARPA 格式的语言模型。可以使用语言模型工具 Kylm (<http://www.phontron.com/kylm/>) 生成出 ARPA 格式的语言模型文件。

(3) 创建发声词典：对于英文可以采用 ARPABET 格式注音的发音词典。由于汉语是由音节 (Syllable) 组成的语言，所以可以采用音节作为汉语语音识别单元。每个音节对应一个汉字，比较容易注音。此外，每个音节由声母和韵母组成，声韵母作为识别单元也是一种选择。

(4) 设置配置文件：在配置文件中设置词典文件和语言模型路径。

(5) 在 eclipse 中执行语音识别的 Java 程序。初始情况下，需要执行 jsapi.exe 或 jsapi.sh 生成出 jsapi.jar 文件。

`edu.cmu.sphinx.tools.feature.FeatureFileDumper` 可以从音频文件导出特征文件, 例如 MFCC 特征。一般提取语音信号的频率特征, 找出语音信号中的音节叫作端点检测, 也就是找出每个字的开始端点和结束端点。因为语音信号中往往存在噪音, 所以不是很容易找准端点。

`Transcriber.jar` 可以实现从声音文件导出讲稿文件。

```
D:\sphinx4-1.0beta5\bin>java -jar -mx300M Transcriber.jar
one zero zero zero one
nine oh two one oh
zero one eight zero three
```

### 5.10.2 视频流内容提取

视频信息一般由四部分组成: 帧、镜头、情节、节目。视频流的内容可以从多个层次分析。

- 底层内容建模, 包括颜色、纹理、形状、空间关系、运动信息等。
- 中层内容建模, 指视频对象 (Video Object), 在 MPEG-4 中包括了视频对象。对象的划分以其独特的纹理、运动、形状、模型和高层语义为依据。
- 高层内容建模 (语义概念等), 例如, 一段体育视频可以提取出扣篮或射门的片断。

关键帧提取策略如下所示。

(1) 设置一个最大关键帧数  $M$ 。

(2) 每个镜头的非边界过渡区的第一帧确定为关键帧。

(3) 使用非极大值抑制法确定镜头边界系数极大值并排序, 以实现基于镜头边界系数的关键帧提取。

镜头边界检测方法有: 只使用镜头边界系数的镜头边界检测——固定阈值法; 结合相邻帧差的镜头边界检测——自适应阈值法。

可以使用 Java 媒体框架 (JMF) API 在 Java 语言中处理声音和视频等时序性的媒体。下面我们首先通过 `vid2jpg.java` 类来提取视频中所有的帧。

```
/**
 * 用这个方法转换从 PushBufferDataSource 发出的数据
```

```

*/
public void transferData(PushBufferStream stream) {
    stream.read(readBuffer);

    //为了防止数据对象中的内容被其他线程修改
    Buffer inBuffer = (Buffer)(readBuffer.clone());

    //检查流是否已经结束
    if(readBuffer.isEOM()) {
        System.out.println("End of stream");
        return;
    }
    //实际把视频中的帧保存到文件
    useFrameData(inBuffer);
}

```

除了关键帧抽取，还可以考虑根据说话人识别和人脸识别来标注或搜索视频。例如，当某人说话的时候，找出视频库里所有该人说的话。对于有声音的视频可以利用视频中的音频信息检索内容。视频流和音频流在时间上是同步的。

手机应用 IntoNow 可以捕捉视频中的声音，通常它只需“听”20 秒左右，就可以判断出你正在看的是什么节目，并列出这个节目的基本信息，以及同样在看这个节目的其他 IntoNow 用户。

## 5.11 内容纠错

输错电话号码，往往只是得到简单的提示“没有这个电话号码”，但是在搜索框中输入错误的搜索词，搜索引擎往往会提示“您是不是要找……”这个正确的词。这个功能也叫“Did you mean”，正式的说法叫作查询纠错（Query correction）或者查询拼写检查。

拼写检查是查询处理极为重要的一个组成部分。在网络搜索引擎用户提交的查询中有 10% 到 15% 的拼写错误，搜索引擎很难用错误拼写的查询词找出相关的文档。拼写检查就是对错误的词给出正确的提示。如果有个正确的词和用户输入的词很近似，则用户的输入可能是错误的。

查询日志中包含大量简单错误的例子，如下所示。



```
poiner sisters -> pointer sisters
brimingham news -> birmingham news
ctamarn sailing -> catamaran sailing
```

类似这些错误可以通过建立正误词表来检查。除此之外，有许多查询日志包含与网站、产品、公司相关的词，对于这样开放类的词不可能在标准的拼写词典中发现。以下是来自同一个查询日志的一些例子。

```
akia 1080i manunal -> akia 1080i manual
ultimatwarcade -> ultimatearcade
mainsourcebank -> mainsource bank
```

因此不存在万能的词表，垂直（网站）搜索引擎往往需要整理和自己行业（网站）相关的词库才能达到好的匹配效果。从搜索日志中挖掘出“错误词->正确词”这样的词对，例如“飞利浦->飞利浦”。

根据正误词表替换用户输入。

```
public static String replace(String content) {
    int len = content.length();
    StringBuilder ret = new StringBuilder(len);
    ErrorDic.PrefixRet matchRet = new ErrorDic.PrefixRet(null, null);

    for(int i=0; i<len;){
        errorDic.checkPrefix(content, i, matchRet); //检查是否存在从当前位置开始的错词
        if(matchRet.value == ErrorDic.Prefix.Match) {
            ret.append(matchRet.data);
            i=matchRet.next; //下一个匹配位置
        }
        else //从下一个字符开始匹配
        {
            ret.append(content.charAt(i));
            ++i;
        }
    }
    return ret.toString();
}
```

因为在各种语言中导致用户输入错误的原因不一样，所以每种语言的正误词的挖掘方式有不一样的地方。对英文单词的搜索需要专门针对英文的拼写检查，对中文词的搜索需要专门针

对中文的拼写检查。

为了讨论对搜索引擎查询最有效的拼写检查技术，首先看下单词拼写检查的概率模型。

$$Spell(w) = \arg \max_{c \in C} P(c | w) = \arg \max_{c \in C} \frac{P(w|c)P(c)}{P(w)}$$

对于任何  $c$  来讲，出现  $w$  的概率  $P(w)$  都是一样的，从而可以忽略它，写成下面的模型。

$$Spell(w) = \arg \max_{c \in C} P(w|c)P(c)$$

这个式子有三个部分： $P(c)$ 、 $P(w|c)$ 、 $\arg \max$ 。

- $P(c)$ ：文章中出现一个正确拼写词  $c$  的概率。也就是说，在英语文章中， $c$  出现的概率有多大呢？因为这个概率完全由英语这种语言决定，一般称之为语言模型。例如，英语中出现  $the$  的概率  $P('the')$  就相对高，而出现  $P('zxzxzxzy')$  的概率接近 0（假设后它也是一个词）。
- $P(w|c)$ ：在用户想键入  $c$  的情况下键入  $w$  的概率。因为这个是代表用户会以多大的概率把  $c$  错输成  $w$ ，因此这个被称为误差模型。
- $\arg \max$ ：用来枚举所有可能的  $c$  并且选取概率最大的那个词。因为有理由相信，一个正确的单词出现的频率高，用户又容易把它写成另一个错误的单词，那么错的单词应该被更正为正确的。

为什么把最简单的一个  $P(c|w)$  变成两项复杂的式子来计算？因为  $P(c|w)$  就是和这两项同时相关的，因此拆成两项反而容易处理。举个例子，一个单词“ $thew$ ”拼错了，看上去“ $thaw$ ”应该是正确的，因为把“ $a$ ”打成“ $e$ ”了。然而，也有可能用户想要的是“ $the$ ”，因为“ $the$ ”是英语中常见的一个词，并且很有可能打字时候手不小心从“ $e$ ”滑到了“ $w$ ”。因此，在这种情况下，想要计算  $P(c|w)$  就必须同时考虑  $c$  出现的概率和从  $c$  到  $w$  的概率。把一项拆成两项让这个问题更加容易、更加清晰。

对于给定词  $w$  可以通过编辑距离挑选出相似的候选正确词  $c$  的集合。编辑距离越小，候选正确词越少，计算也越快。76%的正确词和错误词的编辑距离是 1，所以还需要考虑编辑距离是 2 的情况。99%的正确词和错误词的编辑距离在 2 以内。因此对于拼写检查来说，查找出编辑距离在 2 以内的候选正确词  $c$  的集合就可以了。这是一个模糊匹配的问题。

### 5.11.1 模糊匹配问题

从用户查询词中挖掘正确的提示词表，一般不需要提示没有任何用户搜索过的词。如何从一个大的正确词表中找和输入词编辑距离小于  $k$  的词集合？逐条比较正确词和输入词的编辑距离太慢。

构建一个有限状态自动机准确地识别出和目标词在给定的编辑距离内的字符串的集合。可以输入任何词，然后自动机可以基于是否和目标词的编辑距离最多不超过给定距离，从而接收或拒绝它。

由于 FSA 的内在特性，可以在  $O(n)$  时间内实现， $n$  是测试字符串的长度。而标准的动态规划编辑距离计算方法需要  $O(m*n)$  时间，这里  $m$  和  $n$  是两个输入单词的长度。因此，编辑距离自动机可以更快地检查许多单词和一个目标词是否在给定的最大距离内。

单词“food”的编辑距离自动机形成的非确定有限状态自动机，最大编辑距离是 2。开始状态在左下，状态使用  $n^e$  标记风格命名。这里  $n$  是目前为止正确匹配的字符数， $e$  是错误数量。垂直转换表示未修改的字符，水平转换表示插入，两类对角线转换表示替换（用 \* 标记的转换）和删除（空转换）。

单词“food”的长度是 4，所以图 5-23 有 5 列。允许 2 次错误，所以有 3 行。

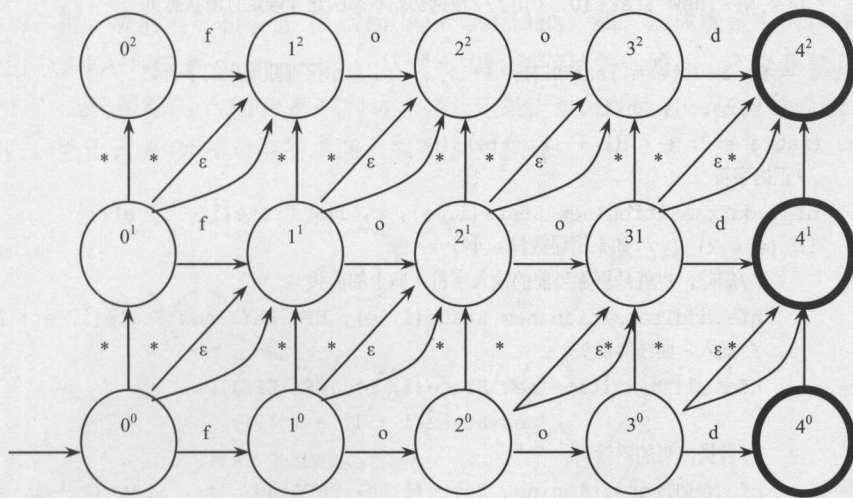


图 5-23 编辑距离自动机



生成编辑距离自动机也就是生成图 5-23。如果不考虑边界节点，一般的节点会发出 4 个状态转换，如图 5-24 所示。

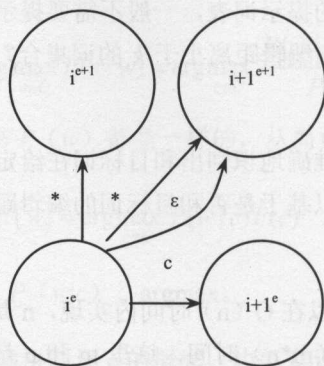


图 5-24 编辑距离自动机中的状态转换

把这里的 \* 和  $\epsilon$  定义成两个特殊的字符。NFA.ANY 表示接收任意字符都可以。NFA.EPSILON 表示不用接收任何字符，可以自动转移到下一个状态。

编辑距离自动机实现代码如下所示。

```
//k 是允许出错的数量
public static NFA levenshteinAutomata(String term, int k) {
    NFA nfa = new NFA(new State(0, 0)); //根据初始状态构建非确定有限状态机

    for (int i = 0; i < term.length(); ++i) { //i 表示正确匹配上的字符数
        char c = term.charAt(i);
        for (int e = 0; e < (k + 1); ++e) {
            //正确字符
            nfa.addTransition(new State(i, e), c, new State(i + 1, e));
            if (e < k) { //如果错误数量 e 小于 k
                //删除，也就是删除当前的输入字符，向上的箭头
                nfa.addTransition(new State(i, e), NFA.ANY, new State(i, e + 1));
                //插入，曲线斜箭头
                nfa.addTransition(new State(i, e), NFA.EPSILON,
                                   new State(i + 1, e + 1));
                //替换，直的斜箭头
                nfa.addTransition(new State(i, e), NFA.ANY, new State(i + 1, e + 1));
            }
        }
    }
}
```

```

}
for (int e = 0; e < (k + 1); ++e) {
    if (e < k)
        nfa.addTransition(new State(term.length(), e), NFA.ANY,
                           new State(term.length(), e + 1)); //最后一列往上的箭头
    nfa.addFinalState(new State(term.length(), e)); //设置结束状态
}
return nfa;
}

```

76%的正确词和错误词的编辑距离是1。23%的正确词和错误词的编辑距离是2。需要在状态对象中记住接收的字符串和原串有几句错误。

看用户输入的某个单词是否和一个正确的单词相似，也就是看它对应的编辑距离自动机能否接收这个正确单词。

```

//构建编辑距离自动机
NFA levenshteinAutomata = NFA.levenshteinAutomata("foxd",1);
//根据幂集构造转换成确定有限状态机
DFA dfa = levenshteinAutomata.toDFA();
//看单词 food 是否能够被接收
System.out.println(dfa.accept("food"));

```

看某个单词是否和给定单词相似就好像先织个矩形的网，然后用这个网去正确词表中捞相似的正确词。把正确的词典和条件（Levenshtein automata）都表示成确定有限状态机，有可能高效地对两个 DFA 取交集（intersect），从词典中找到满足条件的词。取交集就是步调一致地遍历两个 DFA，仅跟踪两个 DFA 都有的边，并且记录走过来的路径。任何时候两个 DFA 都在结束状态时，输出词典 DFA 对应的单词，如图 5-25 所示。

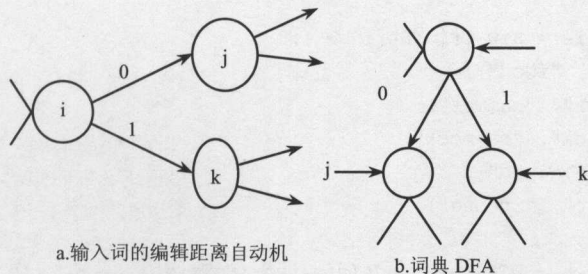


图 5-25 DFA 取交集

```

public static ArrayList<String> intersect(DFA dfal, DFA dfa2) {
    ArrayList<String> match = new ArrayList<String>();//找到的正确单词集合
    Stack<StackValue> stack = new Stack<StackValue>();
    stack.add(new StackValue("", dfal.startState, dfa2.startState));
    while (!stack.isEmpty()) {
        StackValue stackValue = stack.pop();
        Set<Character> ret =
            intersection(dfal.edges(stackValue.s1), dfa2.edges(stackValue.s2));
        for(char edge:ret){
            State statel = dfal.next(stackValue.s1, edge);
            State state2 = dfa2.next(stackValue.s2, edge);
            if(statel!=null&& state2!=null) {
                String prefix = stackValue.s+edge;
                stack.add(new StackValue(prefix, statel, state2));
                if(dfal.isFinal(statel) && dfa2.isFinal(state2)){
                    match.add(prefix);
                }
            }
        }
    }
    return match;
}

```

因为可以把 Trie 树看成确定有限状态机，所以使用标准 Trie 树代替 DFA，标准 Trie 树中存储了正确词库。从正确词库查找相似正确词的整体流程如下所示。

```

//错误词
NFA lev = NFA.levenshteinAutomata("foo",1);
DFA dfa = lev.toDFA();

//正确词表
Trie<String> stringTrie = new Trie<String>();
stringTrie.add("food", "food");
stringTrie.add("hammer", "hammer");
stringTrie.add("hammock", "hammock");
stringTrie.add("ipod", "ipod");
stringTrie.add("iphone", "iphone");
//返回相似的正确词
ArrayList<String> match = DFA.intersect(dfa, stringTrie);

```

现在已经实现了从一个大的正确词表中找和输入词编辑距离小于 k 的词集合，接下来要找出最



大的  $P(c|w)$  对应的  $c$ 。为了计算  $P(c|w)$ ，需要用到  $P(w|c)$  和  $P(c)$ 。根据正确词表中的词频来估计  $P(c)$ ，根据  $w$  和  $c$  之间的编辑距离估计  $P(w|c)$ ，所以实际返回的是如下对象的列表。

```
public class RightWord implements Comparable<RightWord> {
    public String word; //正确词
    public int errors; //错误数，也就是正确词和错误词之间的编辑距离
    public int freq; //在词表中的词频

    public RightWord(String w, int e, int f) {
        word = w;
        errors = e;
        freq = f;
    }

    public int compareTo(RightWord o) { //先比较错误数，再比较词频
        int diff = this.errors - o.errors;
        if(diff!=0) {
            return diff;
        }

        return o.freq - this.freq;
    }
}
```

为了简化选择正确词的计算，可以先把范围缩小到编辑距离小的正确词，对于编辑距离相同的正确词，则取词频高的正确词。

输入“sonly”应该提示“sony”，但是却提示出“only”。对于专有名词可以使用 Jaro-Winkler 距离，因为首字母不容易拼错，所以 Jaro-Winkler 给予了起始部分相同的字符串更高的分数。

```
public class Jaro {
    /**
     *使用 Jaro 距离计算两个字符串的相似性
     *
     * @param string1 第一个输入字符串
     * @param string2 第二个输入字符串
     * @return 一个值在 0~1 之间的相似性
     */
    public float getSimilarity(final String string1, final String string2) {
        //得到字符串取整的一半长度(用于可接受的换位数量的距离)
        final int halflen =
            ((Math.min(string1.length(), string2.length())) / 2)
```

```

        + ((Math.min(string1.length(), string2.length())) % 2);

//得到共同出现的字符
final StringBuffer common1 = getCommonCharacters(string1, string2, halflen);
final StringBuffer common2 = getCommonCharacters(string2, string1, halflen);

//检查是否没有公共字符
if (common1.length() == 0 || common2.length() == 0) {
    return 0.0f;
}

//如果长度不同就返回 0
if (common1.length() != common2.length()) {
    return 0.0f;
}

//得到换位的数量
int transpositions = 0;
int n=common1.length();
for (int i = 0; i < n; i++) {
    if (common1.charAt(i) != common2.charAt(i))
        transpositions++;
}
transpositions /= 2.0f;

//计算 jaro 矩阵
return (common1.length() / ((float) string1.length()) +
        common2.length() / ((float) string2.length()) +
        (common1.length() - transpositions) / ((float) common1.length())) / 3.0f;
}

/**
 * 从 string1 返回在 string2 中的一个字符串缓冲区,
 * 如果它们从在 string1 的位置开始在一个给定距离内
 *
 * @param string1
 * @param string2
 * @param distanceSep
 * @return 字符串缓冲区
 */
private static StringBuffer getCommonCharacters(final String string1,
        final String string2, final int distanceSep) {
    //创建一个字符的返回缓冲区

```

```

final StringBuffer returnCommons = new StringBuffer();
//创建一个用于处理的 string2 的副本
final StringBuffer copy = new StringBuffer(string2);
//遍历 string1
int n=string1.length();
int m=string2.length();
for (int i = 0; i < n; i++) {
    final char ch = string1.charAt(i);
    //如果发现就直接退出循环
    boolean foundIt = false;
    //从一个窗口区间比较字符
    for (int j = Math.max(0, i - distanceSep);
        !foundIt && j < Math.min(i + distanceSep, m - 1); j++) {
        //检查是否发现了
        if (copy.charAt(j) == ch) {
            foundIt = true;
            //追加发现的字符
            returnCommons.append(ch);
            //改变复制的 string2 用于后续处理
            copy.setCharAt(j, (char)0);
        }
    }
}
return returnCommons;
}

```

计算两个词的 Jaro-Winkler 相似度。

```

String s1 ="sony";
String s2 ="sonf";
Jaro j = new Jaro();
System.out.println(j.getSimilarity(s1, s2));

```

### 5.11.2 英文拼写检查

对 101 919 条英文报关公司名统计发现,有拼写出错的为 16 663 条,出错概率为 16.35%。大部分是正确的,如果所有的词都在正确词表中,则不必再查找错误。否则先检查错误词表。最后仍然不确定的,提交查询给搜索引擎,看看是否有错误。

正确词的词典格式是每行一个词,分别是词本身和词频,样例如下所示。



```

biogeochemistry : 1
repairer : 3
wastefulness : 3
battier : 2
awl : 3
preadapts : 1
surprisingly : 3
stuffiest : 3

```

因为互联网中的新词不断出现，正确的词并不是来源于固定的词典，而是来源于搜索的文本本身。直接从文本内容提取英文单词，代码如下所示。不从索引库中提取的原因是 Term 可能经过词干化处理过了，所以用 StandardAnalysis 再次处理。

```

java.io.StringReader input = new java.io.StringReader(content);
TokenStream tokenizer = new StandardTokenizer(input);
for (Token t = tokenizer.next(); t != null; t = tokenizer.next()){
    if( isAllLetter(t.termText()) &&
        (t.termText().length()>=3) &&
        (t.termText().length()<=30) ){
        System.out.println(t.termText());
        fpSource.write(t.termText().toLowerCase());
        fpSource.write(" : 1\n");
    }
}

```

可以根据发音计算用户输入词和正确词表的相似度，还可以根据字面的相似度来判断是否输入错误，并给出正确的单词提示。也可以参考一下开源的拼写检查器（spell checker）的实现，例如，Aspell（<http://aspell.net/>）。

注意，考虑公司名中的拼写错误。一般首字母拼写错误的可能性很小，可以先对名称排序，然后比较前后两个公司名就可以检测出一些非常相似的公司名称了。

另外可以考虑抓取 Google 的拼写检查结果。

```

public static String getGoogleSuggest(String name) throws Exception {
    String searchWord = URLEncoder.encode(name,"utf-8");
    String searchURL = "http://www.google.com/search?q=" + searchWord;
    String strPages=DownloadPage.downloadPage(searchURL);//下载页面

    String suggestWord="";
    Parser parser=new Parser(strPages); //使用 HTMLParser 解析返回的网页

```

```

NodeFilter filter=
    new AndFilter(new TagNameFilter("a"),new HasAttributeFilter("class","spell"));
//取得符合条件的第一个节点
NodeList nodelist=parser.extractAllNodesThatMatch(filter);
int listCount=nodelist.size();

if(listCount>0){
    TagNode node=(TagNode)nodelist.elementAt(0);
    if (node instanceof LinkTag) {</a> 标签
        LinkTag link = (LinkTag) node;
        suggestWord = link.getLinkText();//链接文字
    }
}

return suggestWord;
}

```

### 5.11.3 中文拼写检查

对于汉语的查询纠错，系统预测错误的并不是要产生一个有语法错误的纠错串，而是与原串相比，语义的偏离！怎样才能通过一个带有错误的查询串猜测到用户的真实查询意图呢？给出一个满足条件的纠错串，这个纠错串代表了用户的查询意图，它是查询纠错的目的。语义的偏离才是影响系统性能的首要因素。

和英文拼写检查不一样，中文的用户输入的搜索词串的长度更短，从错误的词猜测可能的正确输入更加困难。这时候需要更多的借助正误词表，词典文本格式如下所示。

```

代款: 贷款
阿地达是: 阿迪达斯
诺基压: 诺基亚
飞利浦: 飞利浦
寂么沙洲冷: 寂寞沙洲冷
欧米加: 欧米茄
欧米枷: 欧米茄
爱力信: 爱立信
西铁成: 西铁城
瑞新: 瑞星
登心绒: 灯心绒

```

在正误词表中，前面一个词是错误的词条，后面是对应的正确词条。为了方便维护，我们还可以把这个词库存放在数据库中。

```
CREATE TABLE CommonMisspellings (
    [misword] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL , --错误词
    [rightword] [varchar] (50) COLLATE Chinese_PRC_CI_AS NULL --正确词
)
```

除了人工整理，还可以从搜索日志中挖掘相似字符串来找出一些可能的正误词对。比较常用的方法是采用编辑距离（Levenshtein Distance）来衡量两个字符串是否相似。编辑距离就是用来计算从原串（s）转换到目标串（t）所需要的最少的插入、删除和替换的数目。例如，源串是“诺基压”，目标串是“诺基亚”，则编辑距离是1。

当一个用户输入错误的查询词没有结果返回时，他可能会知道输入错误，然后用正确的词再次搜索，从日志中能找出来这样的行为，进而找出正确/错误词对。

例如，日志中有如下记录。

```
2007-05-24 00:41:41.0781|DEBUG |221.221.167.147||咯尔喀蒙古|2
...
2007-05-24 00:43:45.7031|DEBUG|221.221.167.147||喀爾喀蒙古|0
...
```

处理每行日志信息，用 `StringTokenizer` 返回“|”分割的字符串，代码如下。

```
StringTokenizer st = new StringTokenizer(line,"|");
while(st.hasMoreTokens()) { //有更多的内容
    System.out.println(st.nextToken()); //取得子串
}
```

假设日志中有搜索结果返回的是正确词，无搜索结果返回的是错误词。挖掘日志的程序如下所示。

```
//存放挖掘的词及搜索出的结果数
HashMap<String,Integer> searchWords = new HashMap<String,Integer>();
while((readline=br.readLine())!=null) {
    StringTokenizer st = new StringTokenizer(readline,"|");
    if(!st.hasMoreTokens()) continue;
    st.nextToken();
    if(!st.hasMoreTokens())continue;
    st.nextToken();
    if(!st.hasMoreTokens())continue;
```



```

st.nextToken();
if(!st.hasMoreTokens())continue;
st.nextToken();
if(!st.hasMoreTokens())continue;
st.nextToken();
if(!st.hasMoreTokens())continue;
//存放搜索词
String key = st.nextToken();
if(key.indexOf(":")>=0) {
    continue;
}
//如果已经处理过这个词就不再处理
if(searchWords.containsKey(key)) {
    continue;
}
if(!st.hasMoreTokens())
{
    continue;
}
String results = st.nextToken();
int resultCount = Integer.parseInt(results);//得到搜索出的结果数

for(Entry<String,Integer> e : searchWords.entrySet()) {
    int diff= Distance.LD(key, e.getKey());
    if(diff ==1 && key.length()>2) {
        if( resultCount == 0 && e.getValue()>0 ) {
            //e.getKey()是正确词, key 是错误词
            System.out.println(key +":"+ e.getKey());
            bw.write(key +":"+ e.getKey()+"\r\n");
        }
        else if(e.getValue()==0 && resultCount>0) {
            //key是正确词, e.getKey()是错误词
            System.out.println(e.getKey() +":"+ key);
            bw.write(e.getKey() +":"+ key+"\r\n");
        }
    }
}
searchWords.put(key, resultCount);//存放当前词及搜索出的结果数
}

```

可以挖掘出如下一些错误/正确词对。

瑜伽服: 瑜伽服

```
落丽塔: 洛丽塔  
巴甫洛: 巴甫洛夫  
hello kiitty: hello kitty
```

除了根据搜索日志挖掘正误词表,还可以根据拼音或字形来挖掘。例如,根据拼音挖掘出“周杰论:周杰伦”,根据字形挖掘出“浙江移动:浙江移动”。搜索“HTMLParser 源代码”时,提示“您是不是要用 SVN 客户端下载?”

## 5.12 术语

Hough Tranform Hough: 变换。

Image Registration: 图像对齐。

moment 矩: 常用的有 Hu 矩和 Zernike 矩。Zernike 矩有旋转不变性、可以缩放、平移不变。

Printed Chinese Character Recognition: 印刷体中文字符识别。

## 5.13 本章小结

本章初步介绍了对各种文档的自动理解。对中文文档来说,首先要避免乱码,然后从文档中提取标题。此外,还可以从文本中提取数字。

NekoHTML 的版本最近一直在更新,Jsoup 的开发也很活跃。除了上述这两个 HTML 解析工具外,常用的还有 HTMLCleaner。和 NekoHTML 类似,HTMLCleaner 支持 XPATH 绝对路径,不过 HTMLCleaner 没有自己的 XPATH 可视化工具。

对 Jsoup 这个名字的理解是——BeautifulSoup for Java,它感觉像 JQuery 和 BeautifulSoup 的混合体。

PDF 是 Adobe 公司在 1993 年开发的电子文件格式,它以独立于操作系统和硬件的方式来表示二维文档格式。在许多操作系统下都有阅读器可以浏览和打印 PDF 格式的文件,例如,Adobe 公司的 Acrobat Reader。PDF 文件格式的完整说明可以从 <http://www.adobe.com/devnet/>

pdf/pdf\_reference.html 网页下载。

和很多大的软件项目一样, PDFBox 一直不太稳定, 开始的时候有中文乱码的问题, 后来又出现过多余空格的错误。

HtmlUnit 调用了 Rhino, 但是 Rhino 速度慢。nashorn 据说是用来代替 Rhino 的。

网页提取的正确率是正确提取的文档数量除以测试集中的文档总数。

除了 JuniversalCharDet, 还有 com.ibm.icu.text.CharsetDetector 可用于二进制流编码检测。

在本章介绍的从各种数据来源提取索引需要的信息, 是爬虫开发中重要而且往往容易碰到问题的部分。各种文档格式处理方式总结如表 5-2 所示。

表 5-2 文档格式处理方式总结

格 式	解 析 包
Microsoft Office OLE2 Compound Document Format(Exce、Word、 PowerPoint、 Visio、 Outlook)	Apache POI
Microsoft Office 2007 OOXML	Apache POI
Adobe Portable Document Format (PDF)	PDFBox
Rich Text Format (RTF)	RtfParser
英文文本	ICU4J
HTML	NekoHTML
XML	Java的javax.xml类
ZIP Archives	Java内部的ZIP类
TAR Archives	Apache Ant
GZIP compression	Java内部的GZIPInputStream
BZIP2 compression	Apache Ant
Image formats (metadata only)	Java的javax.imageio类
Java class files	ASM library (JCR-1522)
Java JAR files	ZIP + Java Class files
MP3 audio	org.farng.mp3
Open Document	直接解析XML
Microsoft Office 2007 XML	Apache POI
MIDI文件	Java内部的javax.sound.midi.*
DWG文件	DWGDirect



# 6

## 第 6 章

### Crawler4j

Crawler4j 是一个容易使用的单机版爬虫软件。

#### 6.1 使用 Crawler4j

可以从 <https://github.com/yasserg/crawler4j> 下载 Crawler4j 的发布 jar 包。相关依赖包在 crawler4j-3.x-dependencies.zip 上。

在一个自己写的控制类中启动抓取任务。WebCrawler 的子类用来定义哪些网站需要遍历，以及页面抓取后怎么处理。

edu.uci.ics.crawler4j.example.simple 包含一个简单的例子，用来定向抓取一个网站。

首先，在 BasicCrawlController 类中增加种子链接，例如，抓取新闻目录页。

```
controller.addSeed("http://roll.news.sina.com.cn/news/gnxw/zs-pl/index.shtml");
```

然后,在 WebCrawler 类的子类中指定定向抓取的遍历规则和页面抓取后的处理方式。例如, BasicCrawler 通过重写 WebCrawler 类中的 shouldVisit 方法判断是否应该遍历指定页面,通过重写 visit 方法来把下载的网页保存到本地硬盘或者数据库中。

```
public class BasicCrawler extends WebCrawler {
    @Override
    public boolean shouldVisit(WebURL url) {
        //根据返回值来指定是否应该遍历这个页面
    }
    @Override
    public void visit(Page page) {
        //网页下载后的处理方法
    }
}
```

在例子中采用如下的条件。

```
private final static Pattern FILTERS = Pattern
    .compile(".*(\\.(css|js|bmp|gif|jpe?g"
        + "|png|tiff?|mid|mp2|mp3|mp4"
        + "|wav|avi|mov|mpe?g|ram|m4v|pdf"
        + "|rm|smil|wmv|swf|wma|zip|rar|gz))$"); //这些文件结尾的不抓
```

运行如下命令启动爬虫。

```
>java BasicCrawlController data 1
```

### 6.1.1 大众点评

抓取大众点评,并保存到数据库。

```
public class Controller {
    public static void main(String[] args) throws Exception {
        String crawlStorageFolder = "/data/crawl/root"; //保存 URL 地址数据的路径
        int numberOfCrawlers = 1; //抓取的线程数量
        CrawlConfig config = new CrawlConfig();
        config.setCrawlStorageFolder(crawlStorageFolder);
        PageFetcher pageFetcher = new PageFetcher(config);
        RobotstxtConfig robotstxtConfig = new RobotstxtConfig();
        RobotstxtServer robotstxtServer = new RobotstxtServer(robotstxtConfig, pageFetcher);
```

```

        CrawlController controller = new CrawlController(config, pageFetcher,
robotstxtServer);
        controller.addSeed("http://www.dianping.com/beijing/food");
        controller.start(MyCrawler.class, numberOfCrawlers);
    }
}

```

一个店铺的 URL 是 <http://www.dianping.com/shop/1848169>，因此，详细页的 URL 模式是 "http://www.dianping.com/shop/\d+"。

<http://www.dianping.com/search/category/2/10/g311> 这样的 URL 里面提取到的饭店信息不全，只有部分信息，因为这是目录页。遍历目录页，然后从目录页得到详细页。目录页是用来找到详细页或者其他目录页的 URL 地址的。先从 <http://www.dianping.com/beijing/food> 这个种子 URL 取得一些目录页面，然后得到详细页面。

对 URL 的处理方式如下所示。

```

public class MyCrawler extends WebCrawler {
    private final static Pattern URLFILTER = Pattern
.compile("http://www.dianping.com/shop/\d+");
    private static int count=0;

    @Override
    public boolean shouldVisit(WebURL url) {
        String href = url.getURL().toLowerCase();
        return URLFILTER.matcher(href).matches();
    }

    @Override
    public void visit(Page page) {
        String url = page.getWebURL().getURL();
        System.out.println("URL: " + url);
        if (page.getParseData() instanceof HtmlParseData) {
            count++;
            if(count%300==0){ //每抓 300 个网页后，休息 10 秒
                Thread.sleep(10000); //休息 10 秒
            }
            if(!url.equals("http://www.dianping.com/beijing/food")){
                HtmlParseData htmlParseData = (HtmlParseData) page.getParseData();
                //用 Jsoup 处理抓下来的网页
            }
        }
    }
}

```



```

    }
}
}
}

```

往数据库插入提取出来的信息。

```

Connection conn = null;
PreparedStatement ps = null;
StringBuffer sql=new StringBuffer("insert into
restaurant(name,valPrice,region,streetaddress,tel,evaluation,description,businesshours,t
ransportation,kouwei,huanjing,fuwu,src,caipinId) values(?,?,?,?,?,?,?,?,?,?,?,?,?)");
try {
    //得到连接查询
    conn = DBConnectionManager.getConnection();
    conn.setAutoCommit(false);
    ps = conn.prepareStatement(sql.toString());
    ps.setString(1, name);
    ps.setString(2, price);
    ps.setString(3, region);
    ps.setString(4, streetAddress);
    ps.setString(5, tel);
    ps.setString(6, evaluation);
    ps.setString(7, description);
    ps.setString(8, businesshours);
    ps.setString(9, transportation);
    ps.setString(10, progressValue.get(0).text());
    ps.setString(11, progressValue.get(1).text());
    ps.setString(12, progressValue.get(2).text());
    ps.setString(13, url);
    ps.setInt(14, caipingId);
    ps.executeUpdate();
} catch (Exception e) {
    try {
        conn.rollback();
    } catch (Exception e1) {
        e1.printStackTrace();
    }
    System.out.println("插入失败");
    e.printStackTrace();
} finally {
    try {

```

```

        conn.setAutoCommit(true);
        ps.close();
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

## 6.1.2 日志

Crawler4j 使用 log4j 记录抓取过程中的日志信息。

```

log4j.rootCategory=DEBUG, stdout

log4j.appender.stdout.Threshold=INFO

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] %m%n

```

## 6.2 crawler4j 原理

直接存储 URL 地址链接太占空间了。使用一个唯一的连续整数的值来标识一个网页，标识网页的整数叫作 docid。

```

public static synchronized int getDocID(String URL) {
    if (there is any key-value pair for key = URL) {
        return value;
    } else {
        lastdocid++;
        put (URL, lastdocid) in storage;
        return lastdocid;
    }
}

```

键/值对存储在 B+树数据结构中。Berkeley DB 作为存储引擎。

it.unimi.dsi.parser 是一个快速轻量级的 HTML 解析器, 它用于提取网页中的网址, 可以从 <http://dsiutils.dsi.unimi.it/> 下载包含这个 HTML 解析器的 jar 包。

对于大网站抓取来说, 爬虫是一个长期运行的后台进程, 可能因为种种原因还没有抓取完, 抓取进程就被中断了。InProcessPagesDB 用于爬虫进程中断后继续抓取任务。

## 6.2.1 代码分析

代码主要分布在 crawler 包和 frontier 包内。

- crawler 包: 管理抓取流程。
- frontier 包: 处理待抓的 URL 地址列表, 管理 url 的唯一编码。

此外, 还有几个辅助包。

- url 包: 用来处理 URL 地址。url.WebURL 代表一条 url, 内含 docid、depth、url 值。url.URLCanonicalizer 将 url 进行 normalize。
- util 包: 包含处理文件和位的工具类。util 包, 用来提供一些工具。

分别说明如下所示。

- Crawler.CrawController: 控制爬虫, 先加种子链接, 再开启多个爬虫, 并且不断监听各个爬虫存活状态。
- Crawler.WebCrawler: 爬虫。
- Run(): 其中是一个轮询 Frontier 的循环, 每次循环从 Frontier 拿 50 条 url, 对每条拿到的 url 调用 preProcessPage(curUrl) 方法。
- processPage(curURL): 用 PageFetcher.fetch 爬取网页, 如果 curURL 有 redirect, 则将 redirect url 的 url 加入 Frontier 以后再调度; 如果爬取正常, 则先进行 parse, 生成 Page, 将新 urls 加入 Frontier (新加入 url 的深度此时确定), 调用 visit(Page){用户自定义操作}。
- Crawler.Configurations: 读取 crawler4j.properties 中的信息。
- Crawler.PageFetcher: 启动 IdleConnectionMonitorThread, 用 fetch(Page, ignoreIfBinary), 爬取单个 Page 页面, 它是一个 static 类。
- Crawler.Page: 一个页面。
- Crawler.PageFetchStatus: 单个页面爬取的配置, 如返回爬取状态数字所代表的含义等。
- Crawler.HTMLParser: 对 HTML 源码进行 parse, 存入 Page 中。



- **Crawler.LinkExtractor**: 抽取出一个 HTML 页面中包含的所有 link。
- **Crawler.IdleConnectionMonitorThread**: 用来监听连接器(用来发送 get 请求, 获取页面), 其 connMgr 则负责 HTML 请求的发送。
- **Init()**: 如果 resumable, 则从 env 所指 home 中读取已处理过的 urls, scheduleAll 加入调度 workQueue 中。
- **Frontier.workQueues**: 要处理的页面集, 如果 resumable, 在构造时会打开对应 env 中的 database(PendingURLsDB), 获取上一次遗留的未处理的 urls。
- **Frontier.inprocessPages**: 当前正在处理的页面集, 继承 workQueues, 存入 InProcessPagesDB 数据库。
- **Frontier.DocIDServer**: 对应数据库 DocIDs, 记录已经见过的页面 url 处理流程: newurl--->workQueues--->inprocessPages--->delete。
- **Robotstxt** 包, 用来判断是否允许抓。

## 6.2.2 使用 Berkeley DB

为了避免重复抓取, URL 地址需要查新。判断解析出的 URL 是否已经遍历过也叫作 URLSeen 测试。URLSeen 测试对爬虫性能有重要的影响。

每个下载下来的网页都有个大于 0 的唯一编号。DocIDServer 中记录已经访问过的 URL 地址列表。DocIDServer.getDocID(url), 如果该 URL 以前见过, 则该值大于 0。

在介绍爬虫架构的时候, 讲解了 Frontier 组件的作用。它作为一个基础的组件, 为爬虫提供 URL。因此, 在 Frontier 中有一个数据结构来存储 URL。在一些小的爬虫程序中, 使用内存队列 (ArrayList、HashMap 或 Queue) 或者优先级队列来存储 URL, 但内存是有限的。在商业应用中, URL 地址数据量非常大。早期的爬虫经常把 URL 地址放在数据库表中, 但数据库对于这种简单的结构化存储来说效率太低。因此, 考虑使用内存数据库来存储。Berkeley DB 就是一种常用的内存数据库。

Berkeley DB (<http://www.oracle.com/database/berkeley-db/index.html>) 是一个嵌入式数据库。这里的嵌入式和嵌入式系统无关, 嵌入式数据库的意思是不需要通过 JDBC 访问数据库, 也不单独启动进程来管理数据。Berkeley DB 运行在网络爬虫所在的进程空间。

Berkeley DB 中的一个数据库只能存储一些键/值对, 也就是键和值两列。Berkeley DB 底层实现采用 B 树, 可以把它看成可以存储大量数据的 HashMap。如果使用 Berkeley DB Java 版本,

则只需要引用一个 jar 包。例如,使用 Berkeley DB Java 的 4.0.103 版本,则在 Eclipse 项目中导入 je-4.0.103.jar。待遍历的 URL 交给 WorkQueues 类管理,它就是用 Berkeley DB 存储待遍历的 URL 地址。

因为 Berkeley DB 的数据库需要在一个环境类的实例中打开,所以首先要实例化一个环境类。

```
//创建一个环境配置对象
EnvironmentConfig envConfig = new EnvironmentConfig();
envConfig.setTransactional(false); //不需要事务功能
envConfig.setAllowCreate(true); //允许创建新的数据库文件
//在路径 envDir 下创建数据库环境
Environment exampleEnv = new Environment(envDir, envConfig);
```

释放环境变量的方法。

```
exampleEnv.sync(); //同步内存中的数据到硬盘
exampleEnv.close(); //关闭环境变量
```

创建数据库,键是字符串,值是一个叫作 WebURL 的类,把 WebURL 保存在 Berkeley DB 中。

```
@Entity
public final class WebURL implements Serializable {
    @PrimaryKey
    private String url;

    private int docid;
}
```

声明数据库配置信息。

```
//创建一个数据库配置对象
DatabaseConfig dbConfig = new DatabaseConfig();
dbConfig.setAllowCreate(true); //允许创建新的数据库文件
dbConfig.setTransactional(false); //不需要事务功能
dbConfig.setDeferredWrite(true); //配置数据库成为延迟写入
```

打开存放要遍历的 URL 的数据库。

```
String databaseName= "PendingURLs";
//用来存储类信息的数据库不要求能够存储重复的关键字
Database pendingURLsDB = env.openDatabase(null, databaseName, dbConfig);
```

```
//声明存储方式
webURLBinding = new WebURLTupleBinding();
```

抓取完毕后要关闭数据库。

```
pendingURLsDB.close();//关闭存储数据库
```

放入一个新发现的 URL 地址。

```
public void put(WebURL curi) throws DatabaseException {
    byte[] keyData = Util.int2ByteArray(curi.getDocid()); //把文档编号转换成字节数组
    DatabaseEntry value = new DatabaseEntry();
    webURLBinding.objectToEntry(cur, value); //把对象转换成数据库中的项目
    pendingURLsDB.put(null, new DatabaseEntry(keyData), value); //放入键/值对
}
```

**WebURLTupleBinding** 用来把一个数据项加工成元组或者反过来，类似于对象的序列化与反序列化。

```
public final class WebURLTupleBinding extends TupleBinding<WebURL> {

    @Override
    public WebURL entryToObject(TupleInput input) {
        //读入网页的 URL 地址和文档编号
        WebURL webURL = new WebURL(input.readString(), input.readInt());
        webURL.setParentDocid(input.readInt()); //来源文档编号
        return webURL;
    }

    //对象存入数据库的格式
    @Override
    public void objectToEntry(WebURL url, TupleOutput output) {
        output.writeString(url.getURL()); //写入 URL 地址
        output.writeInt(url.getDocid()); //写入文档编号
        output.writeInt(url.getParentDocid()); //写入来源文档编号
    }
}
```

为了能正确处理大量数据，要增加 Java 虚拟机运行的内存使用量，否则会出现内存溢出的错误。例如，增加最大内存用量到 800M，可以使用虚拟机参数“-Xmx800m”。

URL 去重利用的是 `DocIDServer.newdocid(url)`，如果该值大于 0，则表示该 URL 以前见过。



通过这个机制，所有以前见过的页面都可以被记录识别。

```
public class DocIDServer {
    protected Database docIDsDB = null;
    protected int lastDocID; //最后一个最大的 URL 文档编号

    public DocIDServer(Environment env) throws DatabaseException {
        DatabaseConfig dbConfig = new DatabaseConfig();
        dbConfig.setAllowCreate(true);
        dbConfig.setTransactional(false);
        dbConfig.setDeferredWrite(true); //延迟写
        docIDsDB = env.openDatabase(null, "DocIDs", dbConfig);
        int docCount = getDocCount();
        if (docCount > 0) {
            //以前的抓取任务中已经保存了 URL 地址
            lastDocID = docCount;
        }
    }
}
```

使用 DocIDServer 实现 URL 判重。

```
EnvironmentConfig envConfig = new EnvironmentConfig();
envConfig.setAllowCreate(true);
envConfig.setTransactional(false);
envConfig.setLocking(true);

Environment env = new Environment(envHome, envConfig);
DocIDServer urlSeen = new DocIDServer(env);

urlSeen.getNewDocID(url); //表示已经处理过这个网页了

if (urlSeen.isSeenBefore(url)){ //看是否已经处理过这个网页
}
```

### 6.2.3 缩短 URL 地址

网上有很多可以在线生成短网址的服务页面，以 <http://gg.gg/> 为例，把网址 “<https://www.snip2code.com/Snippet/52100/An-URL-shortener-algorithm>” 缩短成 “<http://gg.gg/4bjgt>”。

当我们在浏览器里输入 `http://gg.gg/4bjgt` 时，DNS 首先解析短网址，获得 `http://gg.gg` 的 IP 地址。然后，向这个地址发送 HTTP GET 请求，查询 `4bjgt`。`http://gg.gg` 服务器会把请求通过 HTTP 301 转到对应的长 URL “`https://www.snip2code.com/Snippet/52100/An-URL-shortener-algorithm`” 上。

用什么样的算法来缩短一个特定的 URL 呢？在缩短前，先查询这个长地址是否已经缩短过，如果缩短过，只需要返回相应的短地址即可。如果没有则返回一个唯一编号的整数，然后根据这个整数生成缩短 URL 字符串。

这里把缩短 URL 字符串限制为 62 个字母数字字符中的一个，即 26 个字母的大小写加上阿拉伯数字 0 到 9。虽然 -（连字符）和 \_（下划线）允许在 URL 中出现，但是，例如 `http://xyz.com/c0--rw_` 或 `http://xyz.com/_____`，很明显看起来很奇怪。

使用 62 个字符和一个唯一的 7 个长度的字符串，我们可以缩短  $62^7 = 3\,521\,614\,606\,208$  个 URL。

下面是 base10 到 base62 转换器的一个简单的实现代码。

```
public class ShortURLGenerator { //短 URL 地址生成器
    private static final String CHAR_MAP =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    private static final int BASE = CHAR_MAP.length();

    private static String idToShortURL(int id) {
        StringBuilder sb = new StringBuilder();

        while(id > 0) {
            sb.append(CHAR_MAP.charAt(id%BASE));
            id = id / BASE;
        }
        return sb.reverse().toString();
    }

    private static int shortURLtoID(String shortUrl) {
        int no = 0;
        for(char c : shortUrl.toCharArray()) {
            no = no * BASE + CHAR_MAP.indexOf(c);
        }
        return no;
    }
}
```

首先,我们需要在表中查找最后一个唯一的 ID,然后加 1,并将结果数转换为 base62。假设最后一个唯一的 ID 是 678544325,下一个 ID 为 678544326 将被映射到上面的 URL,并且 ID 是 678544326 的 base62 将是:  $45*624+57*623+6*622+23*621+20*620$ 。

表示具有以下数组索引 {45} {57} {6} {23} {20} 的五个字符的 URL。

```
1.String[] elements = {
2."a","b","c","d","e","f","g","h","i","j","k","l","m","n","o",
3."p","q","r","s","t","u","v","w","x","y","z","1","2","3","4",
4."5","6","7","8","9","0","A","B","C","D","E","F","G","H","I",
5."J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X",
6."Y","Z"
7.};
```

得到一个 base62 字符串: JVgxu, 缩短的 URL 是 <http://xyz.com/Jvgxu>。

## 6.2.4 网页编码

Crawler4j 假设所有没有明确声明编码的页面都是采用 UTF-8 字符集编码的。需要增加从二进制流识别字符编码,往往采用 JuniversalCharDet 实现。

## 6.2.5 并发

为抓取的网页生成一个全局唯一的序列号,最容易想到的一个实现方法如下所示。

```
public class UnsafeSequence {
    private int value;

    /** 返回一个唯一的值 */
    public int getNext() {
        return value++;
    }
}
```

如果一个线程调用这个方法,不会有问题。如果多个线程调用这个方法,就会有问题。测试两个线程。

```
public class TestUnsafeSequence extends Thread {
```



```

static UnsafeSequence unsafeSequence = new UnsafeSequence(); //序列号生成器
static HashSet<Integer> seqSet = new HashSet<Integer>(); //已经生成的

public void run() {
    while (true){
        int id = unsafeSequence.getNext();
        if(seqSet.contains(id)){
            System.out.println("序列号重复错误: "+id);
        }
        seqSet.add(id);
    }
}

public static void main(String args[]) {
    (new TestUnsafeSequence()).start();
    (new TestUnsafeSequence()).start();
}
}

```

输出结果如下所示。

```

序列号重复错误: 6503
序列号重复错误: 7582
序列号重复错误: 7849
序列号重复错误: 7971
序列号重复错误: 12599
序列号重复错误: 13175
...

```

返回值重复的原因：因为访问的是同一个 `value` 值。先取得这个值，然后加一。自增操作 `value++` 看起来像是一个单一的操作，但是事实上分为 3 个独立的操作执行它：读取这个值，使之加 1，再写入新值。因为操作发生在多个线程中，这些线程可能交替占有运行时间，所以两个线程很可能同时读取这个值，两个线程都得到相同的值，并都使之增加了 1，结果就是不同的线程返回了相同的序列数。如图 6-1 所示是运气不好的一次执行过程。

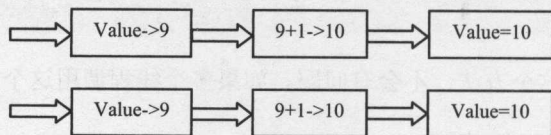


图 6-1 运气不好的一次执行过程

判断 `value` 的值叫竞争条件，避免竞争条件有 3 个方法。

- 无共享：如果能够互不相干的使用，则没问题。例如，数据按类别交给不同的线程处理。
- 使用原子操作：所有对 `value` 值的操作一次性完成，中间不可中断。
- 使用锁：对需要同步的整个方法加锁。

`AtomicInteger` 中的 `incrementAndGet()` 方法是原子性的。这个类位于 `java.util.concurrent.atomic` 包中，用它实现序列号生成器。

```
public class SafeSequence{
    private final AtomicInteger value = new AtomicInteger (0);

    public int getNext() {
        return value.incrementAndGet();//原子性的操作
    }
}
```

用同样的方法测试 `SafeSequence` 中的序列号生成器，测试中没有生成出重复序列号的情况。可以用 `getNext` 声明为 `synchronized` 类型的方法来修正 `UnsafeSequence`，这样就能避免图 6-1 所示的那种不应出现的交互。

```
public class Sequence {
    private int value;

    public synchronized int getNext() {
        return value++;
    }
}
```

`synchronized getNext()` 可以防止多个线程同时访问这个对象的 `getNext` 方法。同时，如果一个对象有多个 `synchronized` 方法，只要一个线程访问了其中的一个 `synchronized` 方法，其它线程不能同时访问这个对象中任何一个 `synchronized` 方法。也就是说，`synchronized` 方法是对象级的锁。这时，不同的对象实例的 `synchronized` 方法是不相干扰的，其它线程照样可以同时访问相同类的另一个对象实例中的 `synchronized` 方法。

除了对象级的锁，还有类级别的锁。

```
public class Sequence {
    private static int value;
```

```

        private static int getNext() {
            synchronized (Sequence.class) {
                return value++;
            }
        }
    }
}

```

这里的 `synchronized` 锁的是一个类。`Sequence.class` 本身是 `Sequence` 类的一个静态属性，也是一个对象。`Sequence.class` 锁的意思就是对整个类加锁，也就是说，无论创建了多少个 `Sequence` 类的对象，这些对象都共享一个相同的锁标记。

`DocIDServer` 类中的 `getDocId` 方法实现了对 `lastDocID` 变量的线程的安全访问，代码如下所示。

```

public class DocIDServer {
    protected int lastDocID;
    protected final Object mutex = new Object();

    public int getDocId(String url) {
        synchronized (mutex) {
            //访问 lastDocID
        }
    }
}

```

## 6.3 本章小结

加利福尼亚大学欧文分校（UCI）的博士生 Yasser Ganjisaffar 开发了 `Crawler4j`。

除了 `Crawler4j`，还有一些流行的开源爬虫，例如，`Nutch`。`Nutch` 不是按网页分抓取任务，而是按域名分的，同一个域名或者 IP 的网页将被分到一个任务里，但是 `Nutch` 爬不了 AJAX。



# 7

## 第 7 章

### 网页排重

不同的网站间转载内容的情况很常见。即使在同一个网站，有时候不同的 URL 地址可能对应同一个页面，或者存在同样的内容以多种方式显示出来，所以，网页需要按内容做文档排重。

例如，一个企业商品搜索。搜商品名，有一家公司发的商品名字都一样，结果这家公司发的商品都显示在前面，但是要求一家企业只显示一条相似的商品在前面，可以把近似重复的文档权重降低，只保留一个文档不降低权重。

判断文档的内容重复有很多种方法，语义指纹的方法比较高效。语义指纹是直接提取一个文档的二进制数组表示的语义，通过比较相等来判断网页是否重复。语义指纹是一个很大的数组，全部存放在内存会导致内存溢出，普通的数据库效率太低，所以采用内存数据库 Berkeley DB。可以通过 Berkeley DB 判断该语义指纹是否已经存在。另外一种方法是通过布隆过滤器来判断语义指纹是否重复。

按词作维度的文档向量维度很高，可以把 SimHash 看成是一种维度削减技术。SimHash 除了可以用在文档排重上，还可以用在任何需要计算文档之间距离的应用上，例如文本分类或聚类。

## 7.1 语义指纹

提取网页语义指纹的方法是：从净化后的网页中，选取最有代表性的一组关键词，并使用该关键词组生成一个语义指纹。通过比较两个网页的语义指纹是否相同来判断两个网页是否相似。

网络上一度出现过很多篇关于“罗玉凤征婚”的新闻报道，其中的两篇新闻内容对比如表 7-1 所示。

表 7-1 两篇新闻内容对比

文档ID	文 档 1	文 档 2
标题	北大清华硕士不嫁的“最牛征婚女”	1米4专科女征婚 求1米8硕士男 应征者如云
内容	24岁的罗玉凤，在上海街头发放了1 300份征婚传单。传单上写了近乎苛刻的条件，要求男方北大或清华硕士，身高1米76至1米83之间，东部沿海户籍。而罗玉凤本人，只有1米46，中文大专学历，重庆綦江人。此事经网络曝光后，引起了很多人的兴趣。“每天都有打电话、发短信求证，或者是应征。”罗玉凤说，她觉得满意的却寥寥无几，“到目前为止只有2个，都还不是特别满意”。	24岁的罗玉凤，在上海街头发放了1 300份征婚传单。传单上写了近乎苛刻的条件，要求男方北大或清华硕士，身高1米76至1米83之间，东部沿海户籍。而罗玉凤本人，只有1米46，中文大专学历，重庆綦江人。此事经网络曝光后，引起了很多人的兴趣。“每天都有打电话、发短信求证，或者是应征。”罗玉凤说，她觉得满意的却寥寥无几，“到目前为止只有2个，都还不是特别满意”。

对于这两篇内容相同的新闻，有可能提取出同样的关键词：“罗玉凤”“征婚”“北大”“清华”“硕士”，这就表示这两篇文档的语义指纹也相同。

为了提高语义指纹的准确性，需要考虑到同义词，例如，“北京华联”和“华联商厦”可以看成相同意义的词。最简单的判断方法是做同义词替换。把“开业之初，比这还要多的质疑的声音环绕在北京华联决策者的周围”替换为“开业之初，比这还要多的质疑的声音环绕在华联商厦决策者的周围”。

设计同义词词典的格式是：每行一个义项，前面是基本词，后面是一个或多个被替换的同义词，请看下面的例子。

华联商厦 北京华联 华联超市

这样可以把“北京华联”或“华联超市”替换成“华联商厦”。对指定文本，要从前往后查找同义词词库中每个要替换的词，然后实施替换。同义词替换的实现代码分为两步。首先是

查找 Trie 树结构的词典过程。

```
public void checkPrefix(String sentence,int offset,PrefixRet ret) {
    if (sentence == null || root == null || "".equals(sentence)) {
        ret.value = Prefix.MisMatch;
        ret.data = null;
        ret.next = offset;
        return ;
    }
    ret.value = Prefix.MisMatch;//初始返回值设为没匹配上任何要替换的词
    TSTNode currentNode = root;
    int charIndex = offset;
    while (true) {
        if (currentNode == null) {
            return;
        }
        int charComp = sentence.charAt(charIndex) - currentNode.splitchar;

        if (charComp == 0) {
            charIndex++;

            if(currentNode.data != null){
                ret.data = currentNode.data;//候选最长匹配词
                ret.value = Prefix.Match;
                ret.next = charIndex;
            }
            if (charIndex == sentence.length()) {
                return; //已经匹配完
            }
            currentNode = currentNode.eqKID;
        } else if (charComp < 0) {
            currentNode = currentNode.loKID;
        } else {
            currentNode = currentNode.hiKID;
        }
    }
}
```

然后是同义词替换过程。

//输入待替换的文本, 返回替换后的文本

```
public static String replace(String content) throws Exception{
```



```

int len = content.length();
StringBuilder ret = new StringBuilder(len);
SynonymDic.PrefixRet matchRet = new SynonymDic.PrefixRet(null,null);

for(int i=0;i<len;){
    //检查是否存在从当前位置开始的同义词
    synonymDic.checkPrefix(content,i,matchRet);
    if(matchRet.value == SynonymDic.Prefix.Match) //如果匹配上,则替换同义词
    {
        ret.append(matchRet.data);//把替换词输出到结果
        i=matchRet.next;//下一个匹配位置
    }
    else //如果没有匹配上,则从下一个字符开始匹配
    {
        ret.append(content.charAt(i));
        ++i;
    }
}

return ret.toString();
}

```

语义指纹生成算法如下所示。

第 1 步：将每个网页分词表示成基于词的特征项，使用  $TF*IDF$  作为每个特征项的权值。地名、专有名词等，名词性的词汇往往有更高的语义权重。

第 2 步：将特征项按照词权值排序。

第 3 步：选取前  $n$  个特征项，然后重新按照字符排序。如果不排序，关键词就找不到对应关系。

第 4 步：调用 MD5 算法，将每个特征项串转化为一个 128 位的串，作为该网页的指纹。

调用 `fseg.result.FingerPrint` 中的方法。

```

String fingerPrint = getFingerPrint("", "昨日，省城濠明北路一名 17 岁的少年在 6 楼晾毛巾时失足坠楼，
摔在楼下的一辆面包车上。面包车受冲击变形时吸收了巨大的反作用力能量，从而“救”了少年一命。目前，伤者尚无生命
危险。据一位目击者介绍，事故发生在下午 2 时 40 分许，当时这名在某美发店工作的少年正站在阳台上晾毛巾，因雨天阳
台湿滑而不小心摔下。记者来到抢救伤者的医院了解到，这名少年名叫李嘉诚，今年 17 岁，系丰城市人。李嘉诚受伤后，
他表姐已赶到医院陪护。据医生介绍，伤者主要伤在头部，具体伤情还有待进一步检查。");
String md5Value = showBytes(getMD5(fingerPrint));
System.out.println("FingerPrint:"+fingerPrint+" md5:"+md5Value);

```

MD5 可以将字符串转化成几乎无冲突的 hash 值，但是 MD5 速度比较慢，MurmurHash 或者 JenkinsHash 也可以生成冲突很少的 hash 值，在 Lucene 的企业搜索软件 Solr1.4 版本中提供了 JenkinsHash 实现的语义指纹，叫作 Lookup3Signature。调用 MurmurHash 生成 64 位的 Hash 值的代码如下所示。

```
public static long stringHash64(String str, int initial) {
    byte[] bytes = str.getBytes();
    return MurmurHash.hash64(bytes, initial);
}
```

## 7.2 SimHash

根据 MD5 方法的语义指纹无法找出特征近似的文档。例如，两个文档相似，但这两个文档的 MD5 值却是完全不同的。关键字的微小差别会导致 MD5 的 hash 值差异巨大，这是 MD5 算法中的雪崩效应（avalanche effect）的结果。输入中一位的变化，散列结果中将有一半以上的位改变。

如果两个相似文档的语义指纹只相差几位或更少，这样的语义指纹叫作 SimHash，可以用海明距离来衡量近似的语义指纹。海明距离是针对长度相同的字符串或二进制数组而言的。对于二进制数组  $s$  和  $t$ ， $H(s, t)$  是两个数组对应位有差别的数量。例如，1011101 和 1001001 的海明距离是 2。下面的方法可以按位比较计算两个 64 位的长整型之间的海明距离。

```
public static int hamming(long l1, long l2) {
    int counter = 0;
    for (int c=0; c<64; C++)
        counter += (l1 & (1L << c)) == (l2 & (1L << c)) ? 0 : 1;
    return counter;
}
```

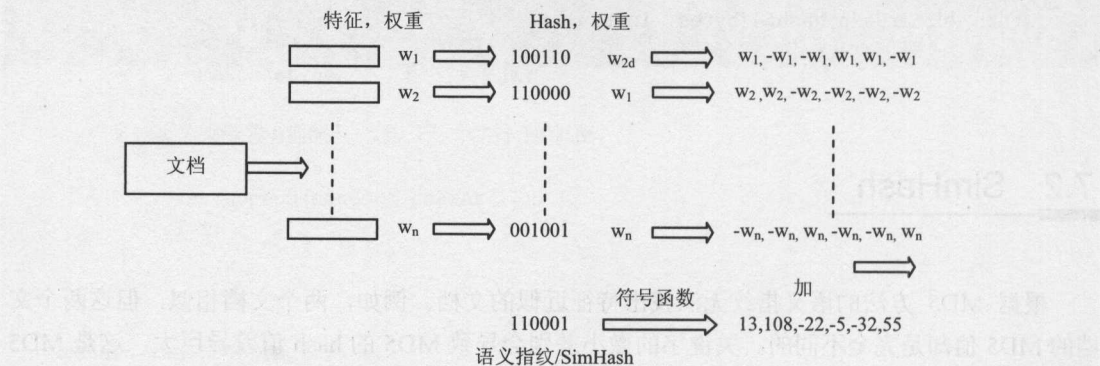
这种按位比较的方法比较慢，可以把两个长整型按位异或（XOR），然后计算结果中 1 的个数，结果就是海明距离。例如计算 A 和 B 两数的海明距离。

```
A = 1 1 1 0
B = 0 1 0 0
A XOR B = 1 0 1 0
```

计算 1010 中 1 的个数是 2，实现代码如下所示。

```
public static int hammingXOR(long l1, long l2) {
    long lxor = l1 ^ l2; //按位异或
    return BitUtil.pop(lxor); //计算1的个数
}
```

假设可以得到文档的一系列的特征，每个特征有不同的重要度。计算文档对应的 SimHash 值的方法是把每个特征的 Hash 值叠加到一起形成一个 SimHash。计算过程如图 7-1 所示。



可以把特征权重看成特征在 SimHash 结果的每一位上的投票权。权重大的特征投票权大，权重小的特征投票权小。所以权重大的特征更有可能影响文档的 SimHash 值中的很多位，而权重小的特征影响文档的 SimHash 值位数很少。

假定 SimHash 的长度为 64 位，文档的 SimHash 计算过程是：首先，初始化长度为 64 的数组，该数组的每个元素都是 0。其次，对于特征列表循环做如下处理。

(1) 取得每个特征的 64 位的 hash 值。

(2) 如果这个 hash 值的第  $i$  位是 1，则将数组的第  $i$  个数加上该特征的权重；反之，如果 hash 值的第  $i$  位是 0，则将数组的第  $i$  个数减去该特征的权重。

最后，完成所有特征的处理，数组中的某些数为正，某些数为负。SimHash 值的每一位与数组中的每个数对应，将正数对应的位设为 1，负数对应的位设为 0，就得到 64 位的 0/1 值的位数组，即最终的 SimHash。

输入特征和权重数组，返回 SimHash 的代码如下所示。



```

public static long simHash(String[] features,int[] weights){
    int[] hist = new int[64]; //创建直方图

    for(int i=0;i<features.length;++i) {
        long addressHash = stringHash(features[i]); //生成特征的 hash 码
        int weight = weights[i];
        /* 更新直方图 */
        for (int c=0; c<64; C++)
            hist[c] += (addressHash & (1 << c)) == 0 ? -weight : weight;
    }

    /* 从直方图计算位向量 */
    long simHash=0;
    for (int c=0; c<64; C++) {
        long t= ((hist[c]>=0)?1:0);
        t <<= c;
        simHash |= t ;
    }

    return simHash;
}

```

要生成好的 SimHash 编码,就要让完全不同的特征差别尽量大,而相似的特征差别比较小。如果特征是枚举类型,只有两个可能的取值,例如, Open 和 Close。Open 返回二进制位全是 1 的散列编码,而 Close 则返回二进制位全是 0 的散列编码。下面的代码将为指定的枚举值生成尽量不一样的散列编码。

```

public static long getSimHash(MatterType matter){
    int b=1; //记录用多少位编码可以表示一个枚举类型的集合
    int x=2;
    while(x<MatterType.values().length) {
        b++;
        x = x<<1;
    }

    long simHash = matter.ordinal();
    int end = 64/b;
    for(int i=0;i<end;++i) {
        simHash = simHash << b; //枚举值按枚举类型总个数向左移位
        simHash += matter.ordinal();
    }
    return simHash;
}

```

## 中文字符串特征的散列算法。

```

public static int byte2int(byte b) {
    return (b & 0xff);
}

private static int MAX_CN_CODE = 6768; //最大中文编码
private static int MAX_CODE = 6768+117; //最大编码

//取得中文字符的散列编码
public static int getHashCode(char c) throws UnsupportedOperationException{
    String s = String.valueOf(c);
    int maxValue = 6768;
    byte[] b = s.getBytes("gb2312");
    if(b.length==2) {
        int index = (byte2int(b[0]) - 176) * 94 + (byte2int(b[1]) - 161);
        return index;
    }
    else if(b.length==1) {
        int index = byte2int(b[0]) - 9 + MAX_CN_CODE;
        return index;
    }
    return c;
}

//取得中文字符串的散列编码
public static long getSimHash(String input) throws UnsupportedOperationException{
    if(input==null || "".equals(input)) {
        return -1;
    }
    int b=13; //记录用多少位编码可以表示一个中文字符

    long simHash = getHashCode(input.charAt(0));
    int maxBit = b;
    for(int i=1;i<input.length();++i) {
        simHash *= MAX_CODE; //把汉字串看成是 MAX_CODE 进制的
        simHash += getHashCode(input.charAt(i));
        maxBit += b;
    }

    long originalValue = simHash;

```

```

for(int i=0;i<=(64/maxBit);++i) {
    simHash = simHash << maxBit;
    simHash += originalValue;
}
return simHash;
}

```

SimHash 的计算依据是要比较的对象的特征,对于结构化的记录可以按列提取特征,而非结构化的文档特征则不明显。如果是新闻,特征可以用标题、最长的几句话。提取特征前,最好先进行一些简单的预处理,如全角转半角。

基于 SimHash 的文档排重流程如图 7-2 所示。

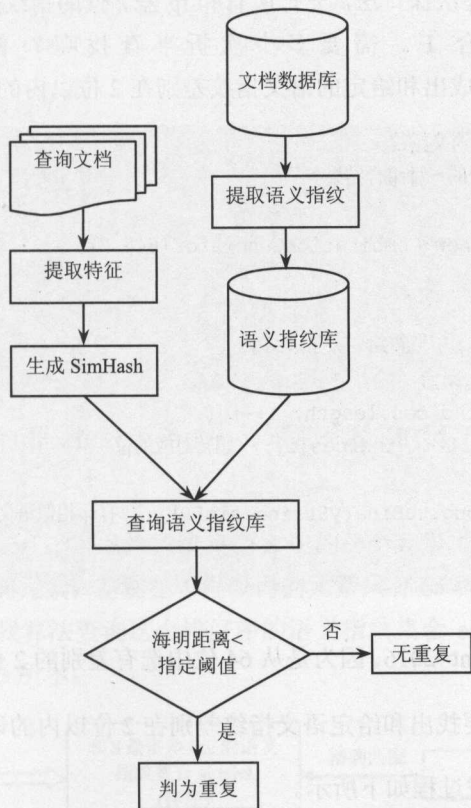


图 7-2 文档排重流程



根据文档排重流程设计出对结构化记录使用排重接口的方式。首先,根据文档集合生成一个语义指纹的集合(fingerprintSet)。然后,根据待查重的文档生成的一个 simHash 值来查找近似的文档集合。fingerprintSet.getSimSet(key, k)返回和 key 相似的数据集合。

```
//返回一条记录的 simHash
long simHashKey = poiSimHash.getHash(poi,address,tel);
//根据一条记录的 simHash 返回相似的数据
HashSet<SimHashData> ret = fingerprintSet.getSimSet(simHashKey, k);
```

把文档转换成 SimHash 后,文档排重就变成了海明距离计算问题。海明距离计算问题是:给出一个  $f$  位的语义指纹集合  $F$  和一个语义指纹  $fg$ ,找出  $F$  中是否存在与  $fg$  只有  $k$  位差异的语义指纹。

最基本的一种方法是逐次探查法,先把所有和  $fg$  差  $k$  位的指纹找出来,然后用折半查找法查找排好序的指纹集合  $F$ 。需要多少次折半查找呢?首先借助组合数生成器 CombinationGenerator 来生成出和给定的语义指纹差别在 2 位以内的语义指纹。

```
long fingerPrint = 1L; //语义指纹
int[] indices; //组合数生成的一种组合结果
//生成差 2 位的语义指纹
CombinationGenerator x = new CombinationGenerator(64, 2);
int count = 0; //计数器
while (x.hasMore()) {
    indices = x.getNext(); //取得组合数生成结果
    long simFP = fingerPrint;
    for (int i = 0; i < indices.length; i++) {
        simFP = simFP ^ 1L << indices[i]; //翻转对应的位
    }
    System.out.println(Long.toBinaryString(simFP)); //打印相似语义指纹
    ++count;
}
```

这里运行的结果是 count=2016。因为是从 64 位中选有差别的 2 位,所以计算公式是  $C_{64}^2 = 64 \times 63 / 2 = 2016$ 。也就是说,要找出和给定语义指纹差别在 2 位以内的语义指纹需要探查 2016 次。

逐次探查法完整的查找过程如下所示。

```
//输入要查找的语义指纹和 k 值,如果找到相似的语义指纹则返回真,否则返回假
public boolean containSim(long fingerPrint,int k) {
```

```

//首先用二分法直接查找语义指纹
if(contains(fingerPrint)) {
    return true;
}

//然后用逐次探查法查找
int[] indices;

for(int ki=1;ki<=k;++ki) {
    //找差1位直到差k位的
    CombinationGenerator x = new CombinationGenerator(64, ki);
    while (x.hasMore()) {
        indices = x.getNext();
        long simFP = fingerPrint;
        for (int i = 0; i < indices.length; i++) {
            simFP = simFP ^ 1L << indices[i];
        }
        //查找相似语义指纹
        if(contains(simFP)) {
            return true;
        }
    }
}

return false;
}

```

在  $k$  值很小，而要找的语义指纹集合  $S$  中的元素不多的情况下，可以用比逐次探查法更快的方法查找。

如果  $k$  值很小，例如  $k=1$ ，可以给指纹集合  $S$  中的每个元素生成出和这个元素差别在 1 位以内的元素。对于长整型的元素，差别在 1 位以内的元素只有 65 种可能。然后把所有新生成的元素排序，最后用折半查找算法查询这个排好序的语义指纹集合  $\text{simSet}$ 。生成法查找近似语义指纹的整体流程，如图 7-3 所示。

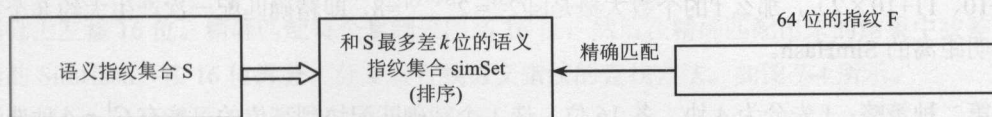


图 7-3 生成法查找近似语义指纹

对给定 SimHash 值  $n$  生成差 1 位的 Hash 值的代码如下所示。

```
for(int j=0;j<64;j++){
long newSimHash=n^1L<<j; //生成和 n 差 1 位的 Hash 值
}
```

在  $k$  值比较小的情况下, 例如  $k \leq 3$ , 介绍另一种快速计算方法。假设有一个已经排序的容量为  $2^d$  的  $f$  位指纹集合。看每个指纹的高  $d$  位, 该高  $d$  位具有以下性质: 因为指纹集合中有很多的位组合存在, 所以高  $d$  位中只有少量重复。

SimHash 排重的假设。

- 整个表中排列组合的部分很少, 不太可能出现例如, 一批 8 位 SimHash, 前 4 位都一样, 但后 4 位出现 16 种 0-1 组合的情况。
- 整个表在前  $d$  位 0-1 分布不会有很多的重复。

这两个假设得到排重的基础: 前  $d$  位上的 0-1 分布足以当成一个指针, 有能力快速搜索前  $d$  位。整个编码空间类似一句古话: 太极生两仪, 两仪生四象, 四象生八卦……

找一个接近于  $d$  的数字  $d'$ , 由于整个表是排好序的, 所以一趟搜索就能找出高  $d'$  位与目标指纹  $F$  相同的指纹集合  $f$ 。因为  $d'$  和  $d$  很接近, 所以找出的集合  $f$  也不会很大。

$$|f| = |s| / 2^{d'}$$

最后在集合  $f$  中查找和  $F$  之间海明距离为  $k$  的指纹也就很快了。核心方法是: 先把检索的集合缩小  $2^{d'}$  倍, 然后在小集合中逐个检查, 看剩下的  $f-d'$  位的海明距离是否满足要求。假设  $k$  值为 3, 有  $2^{34}$  个语义指纹待查找, 则有下面两种策略。

第一种策略:  $f$  分为 6 块, 分别是 11、11、11、11、10、10 位, 最坏的可能是其中 3 块里各出现 1 位海明距离不同, 把这三块限制到低位, 换言之, 选三块精确匹配的到高位, 有  $C_6^3 = 20$  种选法, 因此需要复制 20 个 T 表。对每个表高位做精确匹配, 需要匹配 31~33 位 ( $11 \times 3$ 、 $11 \times 2 + 10$ 、 $11 + 10 \times 2$ ), 那么  $f$  的个数大概是  $|s| / 2^{31} = 2^{[34-31]} = 8$ , 即精确匹配一次产生大约 8 个需要算海明距离的 SimHash。

第二种策略:  $f$  先分为 4 块, 各 16 位, 选 1 个精确匹配块到高位的可能有  $C_4^1 = 4$  种选法。然后对剩下 3 块 48 位切分, 分成 4 块, 各 12 位, 那么选 1 个精确匹配块到高位的可能有  $C_4^1 = 4$



种,  $4 \times 4 = 16$ , 一共要复制 16 次 T 表, 那么高位就有  $16 + 12 = 28$  位。每次精确匹配 28 位后, 产生大约  $2^{[34-28]} = 64$  个需要算海明距离的 SimHash。

假设 SimHash 有  $f$  位, 现在找一个接近于  $d$  的数字  $d'$ , 由于整个表是排好序的, 所以一趟精确匹配就能找出高  $d'$  位与目标指纹  $F$  相同的指纹集合  $f' (f' = 2^{[d-d']})$ 。因为  $d'$  和  $d$  很接近, 所以找出的集合  $f'$  也不会很大。海明距离的比较就在  $f-d'$  位上进行。要确保海明位不同的几位都被限制在  $f-d'$  上, 就需要考虑  $f$  上不同位的组合可能, 即让海明位不会出现在前  $d'$  上, 每种  $f$  位上的组合就需要复制一次 T。

算法本质就是采用分治法, 把问题分解成更小的几个子问题, 降低问题需要处理的数据规模。利用空间 (原空间的  $t$  倍) 和并行计算换时间。分治法查找海明距离在  $k$  以内的语义指纹算法步骤如下所示。

(1) 复制原表 T 为  $T_t$  份:  $T_1$ 、 $T_2$ 、……、 $T_t$ 。

(2) 每个  $T_i$  都关联一个  $pi$  和一个  $\pi_i$ , 其中  $pi$  是一个整数,  $\pi_i$  是一个置换函数, 负责把  $pi$  个 bit 位换到高位上。

(3) 应用置换函数  $\pi_i$  到相应的  $T_i$  表上, 然后对  $T_i$  进行排序。

(4) 对每一个  $T_i$  和要匹配的指纹  $F$ 、海明距离  $k$  做运算: 使用  $F$  的高  $pi$  位检索, 找出  $T_i$  中高  $pi$  位相同的集合, 在检索出的集合中比较剩下的  $f-pi$  位, 找出海明距离小于或等于  $k$  的指纹。

(5) 合并所有  $T_i$  中检索出的结果。

举一个实现的例子, 假设有 100 亿左右 ( $2^{34}$ ) 的语义指纹, SimHash 有 64 位, 所以  $f=64$ ,  $d=34$ , 海明距离  $k$  值是 3。

例如, SimHash 长度是 64 位, 按 16 位拆分, 复制 4 份, 分别是  $T_1$ 、 $T_2$ 、 $T_3$ 、 $T_4$ 。  $T_i$  在  $T_{i-1}$  的基础上左移 16 位。精确匹配每个复制表的高 16 位, 然后在精确匹配出来的结果中找差 3 位以内的 SimHash。按 16 位拆分, 分 4 块查找语义指纹的查找方法, 如图 7-4 所示。

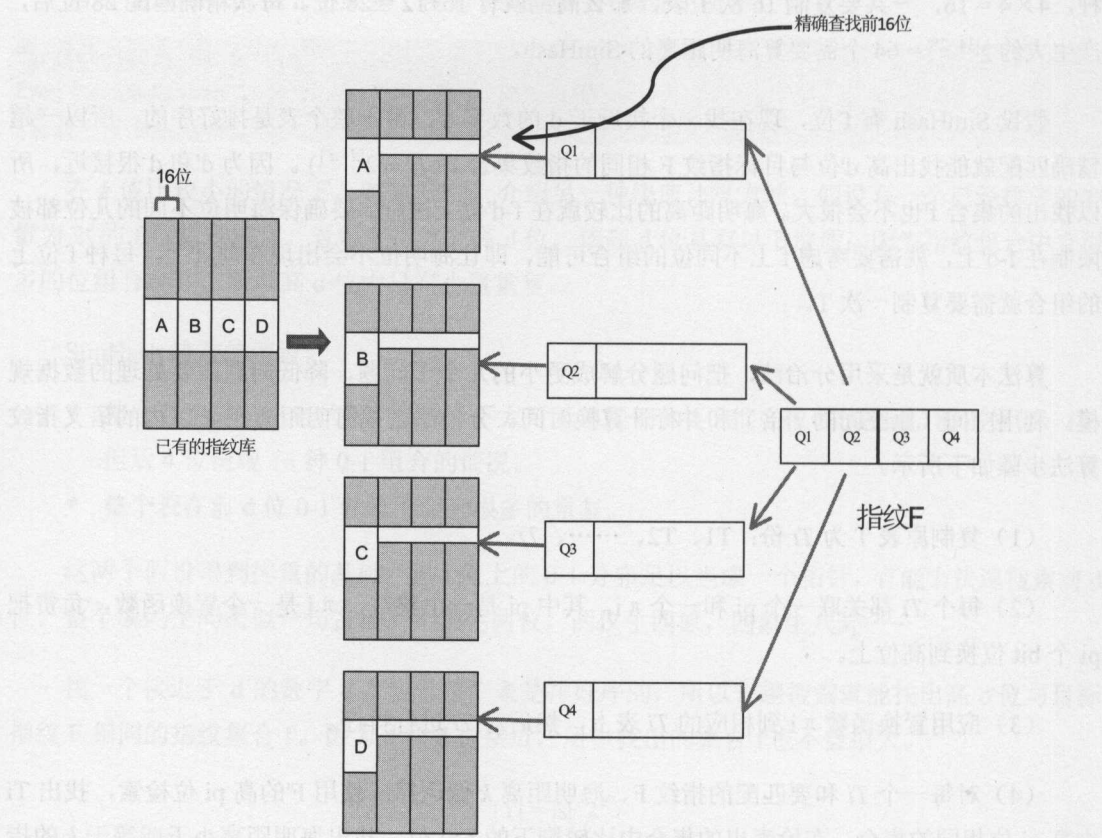


图 7-4 分 4 块查找语义指纹

比较用长整型表示的无符号 64 位语义指纹的代码如下所示。

```
public static boolean isLessThanUnsigned(long n1, long n2) {
    return (n1 < n2) ^ ((n1 < 0) != (n2 < 0)); //无符号比较两个长整数
}

static Comparator<SimHashData> comp = new Comparator<SimHashData>(){
    public int compare(SimHashData o1, SimHashData o2){
        if(o1.q==o2.q) return 0;
        return (isLessThanUnsigned(o1.q,o2.q)) ? 1: -1;
    }
}; //比较无符号 64 位
```

```

static Comparator<Long> compHigh = new Comparator<Long>(){
    public int compare(Long o1, Long o2){
        o1 |= 0xFFFFFFFFFFFFL;
        o2 |= 0xFFFFFFFFFFFFL;
        //System.out.println(Long.toBinaryString(o1));
        //System.out.println(Long.toBinaryString(o2));
        //System.out.println((o1 == o2));
        if(o1.equals(o2)) return 0;
        return (isLessThanUnsigned(o1,o2)) ? 1: -1;
    }
}; //比较无符号 64 位中的高 16 位

public void sort(){//对四个表排序
    t2.clear();
    t3.clear();
    t4.clear();
    for(SimHashData simHash:t1){
        long t = Long.rotateLeft(simHash.q, 16);
        t2.add(new SimHashData(t,simHash.no));

        t = Long.rotateLeft(t, 16);
        t3.add(new SimHashData(t,simHash.no));

        t = Long.rotateLeft(t, 16);
        t4.add(new SimHashData(t,simHash.no));
    }

    Collections.sort(t1, comp);
    Collections.sort(t2, comp);
    Collections.sort(t3, comp);
    Collections.sort(t4, comp);
}

```

## 7.3 分布式文档排重

在批量版本的海明距离问题中，有一批查询语义指纹，而不是一个查询语义指纹。

假设已有的语义指纹库存储在文件 F 中，批量查询语义指纹存储在文件 Q 中。80 亿个 64



位的语义指纹文件 F 是 64GB，压缩后小于 32GB。一批有 1M 大小的语义指纹需要批量查询，因此假设文件 Q 是 8MB。Google 把文件 F 和 Q 存放在 GFS 分布式文件系统。文件分成多个 64MB 的组块，每个组块复制到一个集群中的 3 个随机选择的机器上，每个组块在本地系统存储成文件。

使用 MapReduce 框架，整个计算可以分成两个阶段。在第一阶段，有和 F 的组块数量一样的计算任务（在 MapReduce 术语中，这样的任务叫作 mapper）。

每个任务以整个文件 Q 作为输入在某个 64MB 的组块上求解海明距离问题。

```
public class SimHashMapper extends Mapper<ArrayList<SimHashData>, SimHashSet4, SimHashData,
HashSet<SimHashData>>{
    static int k=3;

    //在 64MB 的组块 fingerprintSet 上求解海明距离问题
    public void map(ArrayList<SimHashData> q,
        SimHashSet4 fingerprintSet,
        Context context) throws IOException, InterruptedException {
        for (SimHashData query : q) {
            HashSet<SimHashData> ret = fingerprintSet.getSimSet(query.q, k);
            if (ret!=null)
                //收集相似的语义指纹集合
                context.write(query, ret);
        }
    }
}
```

一个任务以发现的一个近似重复的语义指纹列表作为输出。在第二阶段，MapReduce 收集所有任务的输出，删除重复发现的语义指纹，产生一个唯一排好序的文件。

```
public class SimHashReducer
    extends
        Reducer<SimHashData, ArrayList<SimHashData>, SimHashData, HashSet<SimHashData>> {
    public void reduce(SimHashData key,
        Iterable<ArrayList<SimHashData>> values,
        Context context) throws IOException, InterruptedException {
        HashSet<SimHashData> dup = new HashSet<SimHashData>();
        for (ArrayList<SimHashData> val : values) {
            dup.addAll(val);
        }
        context.write(key, dup);
    }
}
```

Google 用 200 个任务，扫描组块的合并速度在每秒 1GB 以上。压缩版本的文件 Q 大约是 32GB（压缩前是 64GB），因此总的计算时间少于 100 秒。压缩对于速度的提升起了重要作用，因为对于固定数量的任务，时间与文件 Q 的大小成正比。

通过 Job 类构建一个任务。

```
Job job = new Job(conf, "Find Duplicate");
job.setJarByClass(FindDup.class);
```

## 7.4 本章小结

Broder 提出 shingling 算法用于文档内容相似性检测。Charikar 在论文 *Similarity Estimation Techniques from Rounding Algorithms* 中提出了用 SimHash 实现维度约减。Google 公司的论文 *Detecting NearDuplicates for Web Crawling* 介绍了把 SimHash 用于爬虫抓取过程中的网页去重。本章介绍了用语义指纹排重的基本方法，以及一种特殊的语义指纹 SimHash，详解在 Java 中实现 SimHash 与 SimHash 的压缩存储方法。

# 8

## 第 8 章

### 网页分类

---

用户不太可能输入关键词搜索一个词，更有可能浏览信息分类目录，所以要能准确的对网页分类。

网页可以按功能分类，也可以按内容分类。这里考虑如何按内容自动分类。文本分类程序把一个没见过的文档分成已知类别中的一个或多个，例如，把新闻分成国内新闻和国际新闻。利用文本分类技术可以对网页分类，也可以用于为用户提供个性化新闻或者垃圾邮件过滤。

把给定的文档归到两个类别中的一个叫作两类分类，例如，垃圾邮件过滤只需要确定是不是垃圾邮件。分到多个类别中的一个叫作多类分类，例如，中图法分类目录把图书分成 22 个基本大类。先介绍手工整理规则的关键词加权法分类方法，然后介绍机器学习的方法。



## 8.1 关键词加权法

为了理解机器学习的算法如何对文本分类，首先看一下人是如何对事物分类的。为了判断食物是否是健康食品，可参考食品中的饱和脂肪、胆固醇、糖和钠的含量，把食品分成“健康食品”和“不健康食品”。如果这些值超过一个阈值就认为该食品是“不健康的”，否则是“健康的”。

首先找出一些重要的特征，然后从每个待分类的项目中寻找特征，从抽取出的特征中组合证据（combine evidence），最后根据组合证据按照某种决策机制对项目分类。

在食品分类的例子中，特征是饱和脂肪、胆固醇、糖和钠的含量。可以通过阅读印在食品包装上的营养成分表来取得待分类食品的特征对应的值。为了量化食品的健康程度（记为  $H$ ），有很多方法来组合证据，最简单的方法是按权重求和。

$$H(\text{食物}) = W_{\text{脂肪}} \text{脂肪}(\text{食物}) + W_{\text{胆固醇}} \text{胆固醇}(\text{食物}) + W_{\text{糖}} \text{糖}(\text{食物}) + W_{\text{钠}} \text{钠}(\text{食物})$$

这里  $W_{\text{脂肪}}$ 、 $W_{\text{胆固醇}}$  等是和每个特征关联的重要度。在这个公式里，这些值可能是负数。

在价格搜索中，需要对抓取过来的商品分类。分类的依据是：一些关键词是否在商品标题或者内容列中出现过。

在文本文件中存储关键词类别贡献度，每行的格式如下所示。

**类别名称 关键词 贡献度**

举个例子。

book page 10: 表示类别名称是 book，关键词是 page，贡献度是 10。

book reader 5: 表示类别名称是 book，关键词是 reader，贡献度是 5。

分类的主体过程如下所示。

```
//用一个数组存储一个商品属于某一个类别的隶属度
int degrees = new int[catNum];
```

```

//计算隶属度
for (String content : articals) {
    trie.analysisAll(content, degrees);
}
//商品归类到隶属度最大的类别
int index = 0;
int maxDegree = degrees[0];
for (int i = 1; i < degrees.length; i++) {
    if (maxDegree < degrees[i]) {
        maxDegree = degrees[i];
        index = i;
    }
}
if (maxDegree <= 0) {
    return "";
}
return Categories.catNames[index]; //返回最大隶属度对应的类别

```

写分类规则的方法有：被抓取网站的小类映射到标准的大类。

从字符串中查找英文关键词可以采用标准 Trie 树匹配。

```

//输入内容和隶属度类别数组
public void analysisAll(String prefix,int [] degrees) throws Exception {
    if(rootNode ==null) {
        throw new Exception("rootNode is null");
    }
    int index =0;
    while(index<prefix.length()){
        getMatch(prefix,degrees,index);
        //用 getNextPosition 方法取得下一个匹配点，也就是 index 的值
        index=getNextPosition(index,prefix);
    }
}

//取得内容字符串从 index 位置开始的匹配
public void getMatch(String content,int [] degrees,int index) {
    TrieNode<WordRelation> node = rootNode;
    while(index<content.length()){
        char c =content.charAt(index);
        node = node.getChildren().get(c);
        if (node == null) {//一直到匹配不下去为止
            return;
        }
    }
}

```

```

    }else if(node.isTerminal()){ //匹配上
        WordRelation wr =node.getNodeValue();
        degrees[wr.getIndex()]+=wr.getDegree(); //隶属度增加权重
    }
    index++; //匹配内容字符串的下一个字符
}
}

/**
 * 取得下一个匹配点
 * @param index 当前位置
 * @param prefix 字符串
 * @return 下一个匹配点
 */
public static int getNextPosition(int index,String prefix){
    for(;index<prefix.length();index++){
        char c =prefix.charAt(index);
        if(c=='.'){
            index++;
            while((c>='a' && c<='z') || (c>='A' && c<='Z')){
                index++;
            }
            return index;
        }else if(c==',' ){
            index++;
            while((c>='a' && c<='z') || (c>='A' && c<='Z')){
                index++;
            }
            return index;
        }else if(c==' '){
            index++;
            while((c>='a' && c<='z') || (c>='A' && c<='Z')){
                index++;
            }
            return index;
        }
    }
    return index;
}
}

```

支持中文的文本分类。



```

public final class TernarySearchTrie {
    public class WordRelation {
        public String word; //关键词
        public int[] degree; //关联度, 每个类别都有一个隶属度, 所以是一个数组

        public WordRelation(String word, int[] degree) {
            this.word = word;
            this.degree = degree;
        }
    }

    public final class TSTNode {
        public WordRelation data;

        protected TSTNode paNode;
        protected TSTNode loNode;
        protected TSTNode eqNode;
        protected TSTNode hiNode;

        public char splitter;

        public TSTNode(char key, TSTNode parent) {
            this.splitter = key;
            paNode = parent;
        }

        public String toString() {
            return "data 是" + data + "  splitter是" + splitter;
        }
    }

    public TSTNode rootNode;

    public TernarySearchTrie(String fileName) {
        try {
            InputStream file = new FileInputStream(new File(fileName));
            BufferedReader read = new BufferedReader(new InputStreamReader(
                file, "GBK"));

            String line;
            String cat;

```

```

String word;
String degree;
String[] categories = { "宾馆", "美容", "房产" };
Arrays.sort(categories);
for (int i = 0; i < categories.length; i++) {
    System.err.println(categories[i]);
}
try {
    while ((line = read.readLine()) != null) {

        StringTokenizer st = new StringTokenizer(line, ":");
        while (st.hasMoreElements()) {
            cat = st.nextToken();
            word = st.nextToken();
            degree = st.nextToken();
            int index = Arrays.binarySearch(categories, cat);

            TSTNode currentNode = creatTSTNode(word);
            if (currentNode.data == null) {
                currentNode.data = new WordRelation(word,
                    new int[categories.length]);
            }
            else

                currentNode.data.degree[index] = Integer
                    .parseInt(degree);

        }

    } catch (IOException e) {
        e.printStackTrace();
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
}

//创建一个节点
public TSTNode creatTSTNode(String key) throws NullPointerException,
    IllegalArgumentException {

```



```

    if (key == null) {
        throw new NullPointerException("空指针异常");
    }
    int charIndex = 0;
    if (rootNode == null) {
        rootNode = new TSTNode(key.charAt(0), null);
    }
    TSTNode currentNode = rootNode;
    while (true) {

        int compa = (key.charAt(charIndex) - currentNode.splitter);
        if (compa == 0) {
            charIndex++;
            if (charIndex == key.length()) {

                return currentNode;
            }
            if (currentNode.eqNode == null) {
                currentNode.eqNode = new TSTNode(key.charAt(charIndex),
                    currentNode);
            }
            currentNode = currentNode.eqNode;
        } else if (compa < 0) {
            if (currentNode.loNode == null) {
                currentNode.loNode = new TSTNode(key.charAt(charIndex),
                    currentNode);
            }
            currentNode = currentNode.loNode;
        } else {
            if (currentNode.hiNode == null) {
                currentNode.hiNode = new TSTNode(key.charAt(charIndex),
                    currentNode);
            }
            currentNode = currentNode.hiNode;
        }
    }
}

public WordRelation matchLong(String key, int offset) {

    WordRelation ret = null;
    if (key == null || rootNode == null || "".equals(key)) {
        return null;
    }

```



```

    }

    TSTNode currentNode = rootNode;
    int charIndex = offset;
    while (true) {

        if (currentNode == null) {

            return null;
        }

        int charComp = key.charAt(charIndex) - currentNode.splitter;

        if (charComp == 0) {
            charIndex++;

            if (currentNode.data != null) {

                ret = currentNode.data; //候选最长匹配词
            }

            if (charIndex == key.length()) {
                return ret; //已经匹配完
            }

            currentNode = currentNode.eqNode;
        } else if (charComp < 0) {
            currentNode = currentNode.loNode;
        } else {
            currentNode = currentNode.hiNode;
        }
    }
}
}

```

测试文本分类。

```

TernarySearchTrie dic = new TernarySearchTrie("d:/classfycn.txt");
String[] categories = { "美容", "房产", "宾馆" };
Arrays.sort(categories);
int degrees[] = new int[categories.length];
String sentence = "锦江之星";
int offset = 0;

```

```
WordRelation wr = dic.matchLong(sentence, offset);

for (int i = 0; i < wr.degree.length; i++) {
    degrees[i] = wr.degree[i];
}
//商品归类到隶属度最大的类别
int index = 0;
int maxDegree = degrees[0];
for (int i = 1; i < degrees.length; i++) {
    System.err.println(degrees[i]);
    if (maxDegree < degrees[i]) {
        maxDegree = degrees[i];
        index = i;
    }
}
System.out.println(categories[index]);
```

## 8.2 机器学习的分类方法

文本分类主要分为训练阶段和预测阶段。一个典型的文本分类程序框架，如图 8-1 所示。

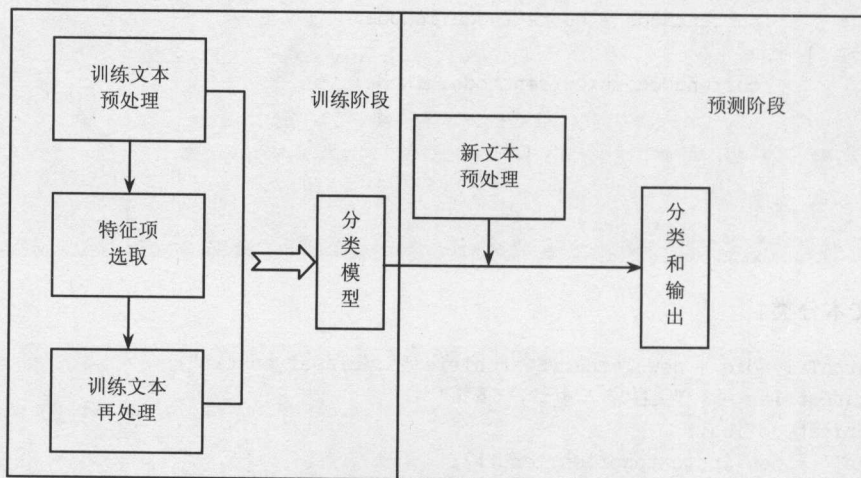


图 8-1 文本分类程序框架

首先准备好训练文本集，也就是一些已经分好类的文本。每个类别路径下包含属于该类别的一些文本文件。例如，文本路径是“D:\train”，类别路径如下。

```
D:\train\政治
D:\train\体育
D:\train\商业
D:\train\艺术
```

“D:\train\体育”类别路径下包含属于“体育”类别的一些文本文件，每个文本文件叫作一个实例(instance)。训练文本文件可以手工整理一些，或者从网络定向抓取网站中已经按栏目分好类的信息。

常见的分类方法有支持向量机、K 个最近的邻居(KNN)和朴素贝叶斯(Naive Bayes)等，可以根据应用场景选择合适的文本分类方法。例如，支持向量机适合对长文本分类，朴素贝叶斯对短文本分类准确度也较高。

为了加快分类的执行速度，可以在训练阶段输出分类模型文件，这样在预测新文本类别阶段就不再需要直接访问训练文本集，只需要读取已经保存在分类模型文件中的信息。可以在预测之前，把分类模型文件预加载到内存中。这里采用单件模式加载分类模型文件。

```
private Classifier() { //读取分类模型
    //...
}
private static Classifier categoryTrie = null;

public static Classifier getInstance() { //取得唯一的实例
    if (categoryTrie == null)
        categoryTrie = new Classifier();
    return categoryTrie;
}
```

依据标题、内容和商品在原有网站的类别把商品归类到新的分类，这些值都是字符串类型。使用参数个数不固定的方式定义分类方法。

```
//根据分类模型分类
public String getCategoryName(String... articals) {
    //获取参数
    for (String content : articals) {
        //根据 content 是否包含某些关键词分类
    }
}
```



```
//返回分类结果
}
```

这样可以按标题或者标题加内容等参数分类。

```
//加载已经训练好的分类模型
Classifier theClassifier = Classifier.getInstance();
//文本分类的内容
String content ="我要买把吉他，希望是二手的，价格2000元以下";
//根据内容分类
String catName = theClassifier.getCategoryName(content);
System.out.println("类别名称:"+catName);
```

交叉验证（Cross Validation）是用来验证分类器的性能的一种统计分析方法。基本思想是把在某意义下将原始数据集进行分组，一部分作为训练集，另一部分作为验证集。首先用训练集对分类器进行训练，再利用验证集来测试训练得到的模型，以此来作为评价分类器的性能指标。

## 8.2.1 特征提取

分类特征是基于词袋模型。不考虑词出现的先后次序。但是词出现的位置可能对意思有影响，例如，没有天花板的舞台，没有舞台的天花板。

待分类的文本往往包括很多单词，很多单词对分类没有太大的贡献，所以需要提取特征词。可以按词性过滤，只选择某些词性作为分类特征，例如，只选择名词和动词作为分类特征词。文本分类的精度随分类特征词的个数持续提高。一般至少可以选 2 000 个分类特征词。

分类特征并不一定就是一个词。例如，可以通过检查标题和签名把文本分类成是否是信件内容。可以看标题是否包含“来自\*\*”和“致\*\*”地址，内容结束处是否包含日期和问候用语等，这样的特征集合仅适用于信件类别。

特征选择的常用方法还有：CHI 方法和信息增益（Information Gain）方法等。首先介绍特征选择的 CHI 方法。

利用 CHI 方法来进行特征抽取基于如下假设：在指定类别文本中出现频率高的词条与在其他类别文本中出现频率比较高的词条，对判定文档是否属于该类别都是很有帮助的。

CHI 方法衡量单词 *term* 和类别 *class* 之间的依赖关系。如果 *term* 和 *class* 是互相独立的，

则该值接近于 0。通过表 8-1 计算一个单词的 CHI 统计变量定义。

表 8-1 CHI 统计变量定义表

	属于class类	不属于class类	合 计
包含单词term	$a$	$b$	$a+b$
不含单词term	$c$	$d$	$c+d$
合计	$a+c$	$b+d$	$a+b+c+d=n$

其中,  $a$  表示属于类别  $class$  的文档集中出现单词  $term$  的文档数;  $b$  表示不属于类别  $class$  的文档集中出现单词  $term$  的文档数;  $c$  表示属于类别  $class$  的文档集中没有出现单词  $term$  的文档数;  $d$  表示不属于类别  $class$  的文档集中没有出现单词  $term$  的文档数;  $n$  代表文档总数。

表 8-1 中单词  $term$  的 CHI 统计公式如下所示。

$$\text{chi\_statistics}(term, class) = \frac{n \times (ad - bc)^2}{(a+c) \times (b+d) \times (a+b) \times (c+d)}$$

类别  $class$  越依赖单词  $term$ , 则 CHI 统计值越大。

表 8-1 也叫作相依表 (contingency table)。开源自然语言处理项目 MinorThird (<http://minorthird.sourceforge.net/>) 中 ContingencyTable 类的 CHI 统计实现代码如下所示。

```
//取对数避免溢出
public double getChiSquared(){
    double n = Math.log(total());
    double num = 2*Math.log(Math.abs((a*d) - (b*c)));
    double den = Math.log(a+b)+Math.log(a+c)+Math.log(c+d)+Math.log(b+d);
    double tmp = n+num-den;
    return Math.exp(tmp);
}
```

计算每个特征对应的 CHI 值。

```
for (Iterator<Feature> i=index.featureIterator(); i.hasNext(); ) { //遍历特征集合
    Feature f = i.next();
    int a = index.size(f,ExampleSchema.POS_CLASS_NAME); //正类中包含特征的文档数
    int b = index.size(f,ExampleSchema.NEG_CLASS_NAME); //负类中包含特征的文档数
    int c = totalPos - a; //正类中不包含特征的文档数
    int d = totalNeg - b; //负类中不包含特征的文档数
```

```
ContingencyTable ct = new ContingencyTable(a,b,c,d);
double chiScore = ct.getChiSquared();//计算特征的 CHI 值
filter.addFeature( chiScore,f );
}
```

如果  $a+c$ 、 $b+d$ 、 $a+b$  或者  $c+d$  中的任意一个值为 0，则会导致除以零溢出的错误。由于数据稀疏导致的问题，可以采用平滑算法来解决。

对所有的候选特征词，按得到的特征区分度排序，如果候选特征词的个数大于 5 000，则选取前 5 000，否则选取所有特征词。

使用 ChiSquareTransformLearner 测试特征提取。

```
Dataset dataset = CnSampleDatasets.sampleData("toy",false);
System.out.println( "old data:\n" + dataset );
ChiSquareTransformLearner learner = new ChiSquareTransformLearner();
ChiSquareInstanceTransform filter =
    (ChiSquareInstanceTransform)learner.batchTrain( dataset );
filter.setNumberOfFeatures(10);
dataset = filter.transform( dataset );
System.out.println( "new data:\n" + dataset );
```

信息增益是广泛使用的特征选择方法。在信息论中，信息增益的概念是：某个特征的值对分类结果的确定程度增加了多少。

信息增益的计算方法是：把文档集合  $D$  看成一个符合某种概率分布的信息源，依靠文档集合的信息熵和文档中词语的条件熵之间信息量的增益关系确定该词语在文本分类中所能提供的信息量。

词语  $w$  的信息量的计算公式如下所示。

$$IG(w)=H(D)-H(D|w)$$

$$=-\sum_{d_i \in D} P(d_i) \times \log_2 P(d_i) + \sum_{w \in \{0,1\}} P(w) \sum_{d_i \in D} P(d_i | w) \times \log_2 P(d_i | w)$$

根据特征在每个类别的出现次数分布计算一个特征的熵。

```
//输入参数 p 是特征的出现次数分布，tot 是特征出现的总次数
public double Entropy(double[] p, double tot){
    double entropy = 0.0;
```



```

for (int i=0; i<p.length; i++) {
    if (p[i]>0.0) { entropy += -p[i]/tot *Math.log(p[i]/tot) /Math.log(2.0); }
}
return entropy;
}

```

计算所有特征的熵。

```

double[] classCnt = new double[ N ];
double totalCnt = 0.0;
for (int c=0; c<N; C++){//循环遍历所有的类别
    classCnt[c] = (double)index.size(schema.getClassName(c)); //每类的文档数
    totalCnt += classCnt[c]; //总文档数
}
double totalEntropy = Entropy(classCnt,totalCnt); //训练文档的总熵值

for (Iterator<Feature> i=index.featureIterator(); i.hasNext(); ) {
    Feature f = i.next();
    double[] featureCntWithF = new double[ N ]; //出现特征的文档在不同类别中的分布
    double[] featureCntWithoutF = new double[ N ]; //不出现特征的文档在不同类别的分布
    double totalCntWithF = 0.0;
    double totalCntWithoutF = 0.0;

    for (int c=0; c<N; C++) {
        featureCntWithF[c] = (double)index.size(f,schema.getClassName(c));
        featureCntWithoutF[c] = classCnt[c] - featureCntWithF[c];
        totalCntWithF += featureCntWithF[c];
        totalCntWithoutF += featureCntWithoutF[c];
    }

    double entropyWithF = Entropy(featureCntWithF,totalCntWithF); //出现特征的熵
    //不出现特征的熵
    double entropyWithoutF = Entropy(featureCntWithoutF,totalCntWithoutF);

    double wf = totalCntWithF /totalCnt; //出现词的概率
    //特征的信息增益
    double infoGain = totalEntropy -wf*entropyWithF -(1.0-wf)*entropyWithoutF;
    igValues.add( new IGPair(infoGain,f) );
}

```

## 8.2.2 朴素贝叶斯

贝叶斯分类的基础是贝叶斯理论 (Bayes' Theorem)。因为贝叶斯理论并不是一个常见的理论, 所以举例说明。

一个医生经常会遇到的问题: 在 40~50 岁这个年龄段参加例行检查的妇女有 1% 的概率有乳腺癌。80% 患有乳腺癌的妇女做乳房透视会得到阳性结果, 9.6% 没有乳腺癌的妇女也会得到阳性的透视结果。一个妇女在这个年龄段例行检查的时候得到阳性的乳房透视结果, 问她确实患有乳腺癌的概率? 如果你以前还没有遇到过这种问题, 在继续阅读之前, 请花点时间提出你自己的答案。通常, 只有约 15% 的医生能得到正确答案。在以上的问题中, 大多数的医生估计她确实患有乳腺癌的可能性会在 70% 到 80% 之间, 这是一个不正确的答案。

把这个问题换一种描述方式, 医生们的回答状况可能会好一些。40~50 岁这个年龄段的妇女接受例行检查, 1 000 人中大概有 10 个人会患有乳腺癌。1 000 个患有乳腺癌的妇女中有 800 人会得到阳性的乳房透视结果。1 000 个没患乳腺癌的妇女中也会有 96 个人得到阳性的乳房透视结果。如果这个年龄段的妇女参加一次例行检查, 得到阳性乳房透视结果的妇女有多少的概率将会真的患有乳腺癌? 这次有 46% 的医生给出正确答案。

40~50 岁这个年龄段参加例行检查的妇女 10 000 个中大概有 100 个会患有乳腺癌。100 个患有乳腺癌的人中有 80 个人会得到阳性的乳房透视结果。9 900 个没有患乳腺癌的人中有 950 个会得到阳性的乳房透视结果。一个得到阳性乳房透视结果的妇女大概有多少概率会真的患有乳腺癌? 正确答案是 7.8%。推导过程: 10 000 个妇女中有 100 个人会患有乳腺癌, 患有乳腺癌的 100 个人中有 80 个人会得到阳性的乳房透视结果。在 10 000 名妇女中有 9 900 个人没患乳腺癌, 这些人中有 950 个人也会得到阳性的乳房透视结果, 这表明得到阳性乳房透视结果的妇女人数是  $(80+950)=1\,030$  人, 1 030 个人中有 80 人患有癌症, 结果为  $(80/1030)=0.07767$  即 7.8%。

换一种方式思考, 在乳房透视筛查之前, 10 000 名妇女可以被分成两组: 第一组, 100 名患乳腺癌的妇女。第二组, 9 900 名不患乳腺癌的妇女。两组相加得到总数 10 000 名患者, 可以确认分组没有把人漏掉。

乳房透视之后, 这些妇女被分成 4 组: A 组: 80 名妇女, 患有癌症并且是阳性的乳房透视。B 组: 20 名妇女, 患有癌症并且是阴性的乳房透视结果。C 组: 950 名妇女, 没患癌症却是阳性的乳房透视结果。D 组: 8 950 名妇女, 没患癌症并且是阴性的乳房透视结果。这 4 组的总和依然是 10 000。

A、B 组的总和，对应患有癌症的第一组；B、C 组的总和对应没有患癌症的第二组。因此乳房透视的结果并没有真正改变患有癌症的人数。癌症病人（A+B）在总患者（A+B+C+D）中的比例跟先前一个妇女患癌症的概率  $(80+20) / (80+20+950+8950) = 1\%$  一样。

通常犯的错误是忽略掉原始的妇女患有癌症的比例，妇女没有乳腺癌收到误报的比例，还有就是把重点仅仅放在患有乳腺癌并且得到阳性乳房透视的妇女比例上。举个例子，大量的医生在这个问题上好像都认为如果 80% 的患有乳腺癌的妇女乳房透视会得到阳性，那么一个妇女乳房透视得到阳性，她患乳腺癌的概率也一定是 80%。

得乳腺癌的病人的原始概率被认为是先验概率（Prior Probability）。患有乳腺癌的病人得到一个乳房透视阳性的概率，和没有患乳腺癌的病人得到一个乳房透视阳性的概率，被认为是条件概率。最终的答案——在乳房透视中呈阳性结果，得出病人得乳腺癌的估计概率，被称为修订概率或者后验概率。表 8-2 是概率变量表。

表 8-2 概率变量表

概 率	值	含 义
$P(\text{癌症})$	0.01	第一组：100个患有乳腺癌的妇女
$P(\sim\text{癌症})$	0.99	第二组：9 900个不患乳腺癌的妇女
$P(\text{阳性} \text{癌症})$	80.0%	80%的患有乳腺癌的妇女在乳房透视化验中得到阳性结果
$P(\sim\text{阳性} \text{癌症})$	20.0%	20%的患有乳腺癌的妇女在乳房透视化验中得到阴性结果
$P(\text{阳性} \sim\text{癌症})$	9.6%	9.6%的没患乳腺癌的妇女在乳房透视化验中得到阳性结果
$P(\sim\text{阳性} \sim\text{癌症})$	90.4%	90.4%的没患乳腺癌的妇女在乳房透视化验中得到阴性结果
$P(\text{癌症}\&\text{阳性})$	0.008	A组：80个患有乳腺癌的妇女并且在乳房透视化验中得到阳性结果
$P(\text{癌症}\&\sim\text{阳性})$	0.002	B组：20个患有乳腺癌的妇女并且在乳房透视化验中得到阴性结果
$P(\sim\text{癌症}\&\text{阳性})$	0.095	C组：950个没有患乳腺癌的妇女并且在乳房透视化验中得到阳性结果
$P(\sim\text{癌症}\&\sim\text{阳性})$	0.895	D组：8 950个没患乳腺癌的妇女并且在乳房透视化验中得到阴性结果
$P(\text{阳性})$	0.103	1 030个得到阳性结果的妇女
$P(\sim\text{阳性})$	0.897	8 970个得到阴性结果的妇女
$P(\text{癌症} \text{阳性})$	7.80%	如果乳房透视化验是阳性，那么患有乳腺癌的概率为7.8%
$P(\sim\text{癌症} \text{阳性})$	92.20%	如果乳房透视化验是阳性，那么健康的概率为92.2%
$P(\text{癌症} \sim\text{阳性})$	0.22%	如果乳房透视化验为阴性，那么患乳腺癌的概率为0.22%
$P(\sim\text{癌症} \sim\text{阳性})$	99.78%	如果乳房透视化验中得到阴性结果，那么健康的概率为99.78%

乳房透视结果为阳性的妇女患有乳腺癌的概率的计算公式如下所示。



$$\begin{aligned}
 & \frac{P(\text{阳性} | \text{癌症}) * P(\text{癌症})}{P(\text{阳性} | \text{癌症}) * P(\text{癌症}) + P(\text{阳性} | \sim \text{癌症}) * P(\sim \text{癌症})} \\
 &= \frac{P(\text{阳性} \& \text{癌症})}{P(\text{阳性} \& \text{癌症}) + P(\text{阳性} \& \sim \text{癌症})} \\
 &= \frac{P(\text{阳性} \& \text{癌症})}{P(\text{阳性})} \\
 &= P(\text{癌症} | \text{阳性})
 \end{aligned}$$

这个计算的完整的一般形式被称为贝叶斯定理或贝叶斯规则，如下所示。

$$P(A | X) = \frac{P(X | A) * P(A)}{P(X | A) * P(A) + P(X | \sim A) * P(\sim A)}$$

贝叶斯定理的推理过程如下所示。

$$\begin{aligned}
 P(A | X) &= \frac{P(X \& A)}{P(X)} \\
 &= \frac{P(X \& A)}{P(X \& A) + P(X \& \sim A)} \\
 &= \frac{P(X | A) * P(A)}{P(X | A) * P(A) + P(X | \sim A) * P(\sim A)}
 \end{aligned}$$

由  $P(A|X)$  得到  $P(X\&A)/P(X)$ ，这看起来相是一个恒真命题，但是在实际上数学的表现却是不同的。 $P(A|X)$  是一个单一的数字，是  $A$  在  $X$  子集中的一个归一化的概率。 $P(X\&A)/P(X)$  是在全部的样本中  $X\&A$  和  $X$  出现的频率。 $P(\text{癌症}|\text{阳性})$  是一个百分比或者说是一个概率，取值范围在 0 到 1 之间。 $(\text{阳性}\&\text{癌症})/(\text{阳性})$  既可以用概率来衡量，例如  $0.008/0.103$ ，也可以表示为妇女群体，例如  $194/2494$ 。分母从  $P(X)$  转换到  $P(X\&A)+P(X\&\sim A)$  是一个非常简单的步骤，它的主要目的是作为一个中间步骤到达最终的等式。得到贝叶斯定理的最后一步是将分子和分母中的  $P(X\&A)$  转换成  $P(X|A)*P(A)$ ，分母中的  $P(X\&\sim A)$  转换成  $P(X|\sim A)*P(\sim A)$ 。

贝叶斯理论的一般形式如下所示。

$$P(C | D) = \frac{P(D | C) P(C)}{P(D)} = \frac{P(D | C) P(C)}{\sum_{c \in C} P(D | C = c) P(C = c)}$$

其中  $C$  和  $D$  是随机变量。

贝叶斯分类公式如下所示。

$$\text{Class}(d) = \arg \max_{c \in C} P(c|d) = \arg \max_{c \in C} \frac{P(d|c)P(c)}{\sum_{c \in C} P(d|c)P(c)}$$

其含义是，在所有可能的类集合  $C$  中返回类  $c$ ，使得  $P(c|d)$  最大。这里并不直接计算  $P(c|d)$ ，根据贝叶斯理论，可以通过计算  $P(d|c)$  和  $P(c)$  得到  $P(c|d)$ 。 $P(c)$  是先验概率，而  $P(d|c)$  是类别  $c$  产生出文档  $d$  的条件概率。

为了简化计算过程，朴素贝叶斯模型假定特征变量相互独立，也就是待分类文本中的词与词之间没有关联。假设  $d=w_1, w_2, \dots, w_n$ ，则  $P(d|c) = \prod_{i=1}^n P(w_i|c)$

给定一个类  $c$ ，我们为每个词定义一个布尔型的随机变量  $w_i$ 。布尔型事件的结果是 0 或 1。 $P(w_i=1|c)$  的概率是项  $w_i$  通过类  $c$  产生的概率。相反地， $P(w_i=0|c)$  的概率可以说是项  $w_i$  不通过类  $c$  产生的概率。这是多变量伯努利 (Multiple Bernoulli) 事件空间。

在这个事件空间下，为每个项在某些类  $c$  下，估计这个词是由这个类生成的可能性。例如，在垃圾分类中， $P(\text{cheap}=1|\text{spam})$  可能有很高的概率，但是  $P(\text{dinner}=1|\text{spam})$  将有一个很低的概率。

表 8-3 在多变量伯努利事件空间中如何表示文档

文档id	cheap	buy	banking	dinner	the	class
1	0	0	0	0	1	not spam
2	1	0	1	0	1	spam
3	0	0	0	0	1	not spam
4	1	0	1	0	1	spam
5	1	1	0	0	1	spam
6	0	0	1	0	1	not spam
7	0	1	1	0	1	not spam
8	0	1	0	0	1	not spam
9	0	0	0	0	1	not spam
10	1	0	0	1	1	not spam

表 8-3 显示怎样设置训练文档能被呈现在这个事件空间中。在这个例子中有 10 个文档，每个文档用唯一的 id 标识，两类 (spam 和 not spam) 和包含 “cheap” “buy” “banking” “dinner”

“the” 项的词汇表。在这个例子中  $P(\text{spam})=3/10$ ,  $P(\text{not spam})=7/10$ 。接着, 估计每个词和类搭配的  $P(w|c)$ 。最直接的方法是使用最大似然法估计概率, 公式如下所示。

$$P(w|c) = \frac{df_{w,c}}{N_c}$$

这里,  $df_{w,c}$  是在类  $c$  中包含词  $w$  的训练文档的数量,  $N_c$  是类  $c$  的训练文档的总数。最大似然估计无非是在类别  $c$  中包含项  $w$  的文档的比例。使用最大似然估计, 容易计算  $P(\text{the}|\text{spam})=1$ ,  $P(\text{the}|\text{not spam})=1$ ,  $P(\text{dinner}|\text{spam})=0$ ,  $P(\text{dinner}|\text{not spam})=1/7$  等。

使用多变量伯努利模型, 文档似然值  $P(d|c)$  如下所示。

$$P(d|c) = \prod_{w \in V} P(w|c)^{\delta(w,d)} (1-P(w|c))^{1-\delta(w,d)}$$

其中当且仅当在文档  $d$  中出现词  $w$  时,  $\delta(w,d)$  是 1。

实际上, 由于 0 概率问题, 所以不可能使用最大似然估计。为了解释 0 概率问题, 让我们回到表 8-3 中的垃圾邮件分类的例子。假设我们接收一封垃圾邮件包含“dinner”一词。无论电子邮件包含或不包含其他的词,  $P(d|c)$  一直是 0, 因为  $P(\text{dinner}|\text{spam})=0$ , 而这个词在文档中出现 (也就是  $\delta(\text{dinner}, d)=1$ )。因此, 任何包含词“dinner”的文档都会自动计算成是垃圾的概率是零。这个问题比较普遍, 因为每当一个文档包含一个词, 而那个词从来不出现在一个或多个类时, 零概率问题就会产生。这里的问题是最大似然估计是基于训练集中的出现计数。然而, 这个训练集是有限的, 因此不可能观察到所有可能的事件, 这就是所谓的数据稀疏。稀疏往往是因为训练集太小, 但是也会发生在比较大的数据集上。因此, 必须改变这样一种方式的估计, 对所有词, 包括那些没有在给定类中观察到的, 给予一些概率量。我们必须为所有在词典中的项确保  $P(w|c)$  是非零的。通过这样做, 就能避免所有的与零概率相关的问题。平滑技术可以克服零概率问题, 一个流行的平滑技术是贝叶斯平滑。贝叶斯平滑是假设模型上的某些先验概率, 并使用最大后验估计 (max a posteriori)。由此产生了平滑估计的多变量伯努利模型的形式。

$$P(w|c) = \frac{df_{w,c} + \alpha_w}{N_c + \alpha_w + \beta_w}$$

$\alpha_w$  和  $\beta_w$  是依赖于  $w$  的参数。不同的参数设置导致不同的估计结果。一种流行的选择是对所有  $w$  设置  $\alpha_w = 1$  和  $\beta_w = 0$ , 结果是以下的估计。



$$P(w|c) = \frac{df_{w,c} + 1}{N_c + 1}$$

另一个选择是，对所有的  $w$  设置  $\alpha_w = \mu \frac{N_w}{N}$  和  $\beta_w = \mu(1 - \frac{N_w}{N})$ 。其中  $N_w$  是包含词  $w$  的训练文档总数， $\mu$  是可调参数，结果如下所示。

$$P(w|c) = \frac{df_{w,c} + \mu \frac{N_w}{N}}{N_c + \mu}$$

此事件空间只记录了一个词是否出现，而没有记录这个词出现了多少次，但词频可是一个重要的信息，对长文本来说尤其如此。现在描述一个考虑到频率的多项（multinomial）事件空间。

表 8-4 在多项事件空间中如何表示文档

文档id	cheap	buy	banking	dinner	the	class
1	0	0	0	0	2	not spam
2	3	0	1	0	1	spam
3	0	0	0	0	1	not spam
4	2	0	3	0	2	spam
5	5	2	0	0	1	spam
6	0	0	1	0	1	not spam
7	0	1	1	0	1	not spam
8	0	1	0	0	1	not spam
9	0	0	0	0	1	not spam
10	1	0	0	1	1	not spam

在表 8-4 的例子中有 10 个文档（每个文档用唯一的 id 标识），两类（spam 和 not spam）和包含“cheap”“buy”“banking”“dinner”“the”项的词汇表。和多变量伯努利表示唯一的区别是，事件不再是布尔型的。多项模型的最大似然估计和多变量伯努利模型很相似，公式如下所示。

$$P(w|c) = \frac{tf_{w,c}}{|c|}$$

$tf_{w,c}$  是训练集中的词  $w$  在类  $c$  中出现的次数，而  $|c|$  是属于类  $c$  的词总次数。在垃圾分类例子中， $P(\text{the}|\text{spam})=4/20$ ， $P(\text{the}|\text{not spam})=9/15$ ， $P(\text{dinner}|\text{spam})=0$ ，而  $P(\text{dinner}|\text{not spam})=1/15$ 。

因为词是多项分布，所以给定类  $c$  的文档  $d$  的似然公式如下所示。

$$P(d|c) = P(|d|)(\text{tf}_{w1,d}, \text{tf}_{w2,d}, \dots, \text{tf}_{wv,d})! \prod_{w \in V} P(w|c)^{\text{tf}_{w,d}}$$

$\text{tf}_{w,d}$  是词  $w$  在文档  $d$  中出现的次数，而  $|d|$  是出现在文档  $d$  中的词总次数。 $P(|d|)$  是产生长度是  $|d|$  的文档的概率，而  $(\text{tf}_{w1,d}, \text{tf}_{w2,d}, \dots, \text{tf}_{wv,d})!$  是多项系数。注意， $P(|d|)$  和多项系数是依赖于文档的，为了分类的目的，可以省略掉这两项。因此实际需要计算的是  $\prod_{w \in V} P(w|c)^{\text{tf}_{w,d}}$ 。

词似然值的贝叶斯平滑估计根据如下公式计算。

$$P(w|c) = \frac{\text{tf}_{w,c} + \alpha_w}{|c| + \sum_{w \in V} \alpha_w}$$

$\alpha_w$  是一个依赖于  $w$  的参数。对所有的  $w$ ，设置  $\alpha_w = 1$  是一种可能的选择。对应如下的估计。

$$P(w|c) = \frac{\text{tf}_{w,c} + 1}{|c| + |V|}$$

另一个流行的选择是，设置  $\alpha_w = \mu \frac{cf_w}{|c|}$ ，这里  $cf_w$  是词  $w$  出现在训练文档中的总次数。 $|c|$  是词在所有的训练文档中出现的总次数，而  $\mu$  则是一个可调的参数。在这个设置下，得到如下的估计。

$$p(w|c) = \frac{\text{tf}_{w,c} + \mu \frac{cf_w}{|c|}}{|c| + \mu}$$

多变量伯努利模型又叫文档型模型。多项模型又叫词频型模型。这里实现文档型分类模型。

计算先验概率的实现代码如下所示。

```
private static TrainingData trainingData=TrainingData.getInstance();//得到训练集

/**
 * 先验概率
 * @param c 给定的分类
```

```

* @return 给定条件下的先验概率
*/
public static float calculatePc(String c){
    float Nc = trainingData.getClassDocNum(c); //给定分类的训练文本数
    float N = trainingData.getTotalNum(); //训练集中文本总数
    return (Nc / N);
}

```

计算类条件概率，类条件概率的公式如下所示。

$$P(w_i | c_j) = \frac{N(W=w_i, C=c_j)+1}{N(C=c_j)+M+V}$$

其中， $N(W=w_i, C=c_j)$ 表示类别  $c_j$  中包含词  $w_i$  的训练文本数量； $N(C=c_j)$  表示类别  $c_j$  中的训练文本数量； $M$  值用于平滑，避免  $N(W=w_i, C=c_j)$  过小所引发的问题； $V$  表示类别的总数。

```

/**
 * 计算类条件概率
 * @param w 给定的词
 * @param c 给定的分类
 * @return 给定条件下的类条件概率
 */
public static float calculatePwc(String w, String c) {
    //返回给定分类中包含分类特征词的训练文本的数目
    float dfwc = tdm.getCountContainKeyOfClassification(c, w);
    //返回训练文本集中在给定分类下的训练文本数目
    float Nc = tdm.getClassDocNum(c);
    //类别数量
    float V = tdm.getTraningClassifications().length;
    return ( (dfwc + 1) / (Nc + M + V) );
}

```

利用样本数据集计算先验概率和各个文本向量属性在分类中的条件概率，从而计算出各个概率值，最后比较各个概率值，选出最大的概率值对应的文本类别，即为文本所属的分类。

为了避免结果过小，对乘积的结果取对数，因此计算整个概率的公式如下所示。

$$\log(P(d|c)) + \log(P(c))$$



```

/**
 * @param d 给定文本的属性向量
 * @param Cj 给定的类别
 * @return 类别概率
 */
float calcProd(String[] d, String Cj){
    float ret = 0.0F;
    //类条件概率连乘
    for (int i = 0; i < d.length; i++){
        String wi = d[i];
        //因为连乘结果过小, 所以转成取对数
        //对分类的最终结果无影响, 因为只是比较概率大小而已
        ret+=Math.log(ClassConditionalProbability.calculatePwc(wi, Cj));
    }
    //再乘以先验概率
    ret += Math.log(PriorProbability.calculatePc(Cj));
    return ret;
}

```

最后取最大的概率对应的类。

```

String[] classes = tdm.getTraningClassifications();//返回所有的类名
float probability = 0.0F;
List<ClassifyResult> crs = new ArrayList<ClassifyResult>();//分类结果
for (int i = 0; i < classes.length; i++){
    String ci = classes[i];//第 i 个分类
    //计算给定的文本属性向量 terms 在给定的分类 Ci 中的分类条件概率
    probability = calcProd(terms, ci);
    //保存分类结果
    ClassifyResult cr = new ClassifyResult(ci,probability);
    crs.add(cr);
}
//返回概率最大的分类
float maxPro = crs.get(0).probability;
String c = crs.get(0).classification;
for(ClassifyResult cr:crs){
    if(cr.probability>maxPro){
        c = cr.classification;
        maxPro = cr.probability;
    }
}
return c;

```

## 8.2.3 支持向量机

和贝叶斯分类器基于概率来分类不同，支持向量机是基于几何学原理来实现分类的。假设“+”是特征空间的一类点，而“-”是特征空间中的另外一类点。SVM通过分类面来判断特征空间中的待分类点的类别。分类间隔的基本思想可用图8-2的两维情况说明。

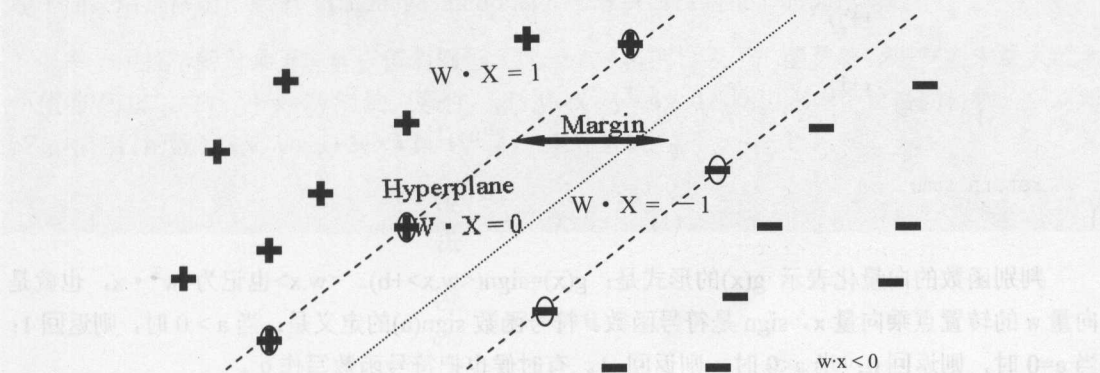


图 8-2 分类间隔

假设有  $N$  个点在  $p$  维特征空间  $X = \{x_1, x_2, \dots, x_p\}$  中分别属于两类  $C_+$  和  $C_-$ 。要解决的问题是找到一个函数  $f(x_1, x_2, \dots, x_p)$  判别出两类，对于  $C_+$  类的点返回正值，而对于  $C_-$  类的点返回负值。这个函数叫作判别函数（discriminant function）。

如果判别函数是线性的，则可以把判别函数看成  $f(x_1, x_2, \dots, x_p) = w_1 * x_1 + w_2 * x_2 + \dots + w_p * x_p + b$ 。

假设向量  $w = (w_1, w_2, \dots, w_p)$ ，向量  $x = (x_1, x_2, \dots, x_p)$ ，用  $\langle w, x \rangle$  表示向量  $w$  和  $x$  之间的内积（inner product），或者叫作点积。

为了快速计算两个数组  $x$  和  $y$  的点积，只计算相同位置不为 0 的值就好了。对于稀疏维度，为了节省空间  $\langle \text{Index}, \text{Value} \rangle$  不采用哈希表储存，而是将 index 全部按升序排列，然后通过归并两个排好序的数组来计算。有点类似搜索引擎中的 docList 集合求 AND 的操作。计算的时间复杂度是  $O(m+n)$ 。计算点积的实现代码如下所示。

```
static double scalarProduct(Node[] x, Node[] y){
    double sum = 0;
    int xlen = x.length;
```

```

int ylen = y.length;
int i = 0;
int j = 0;
while(i < xlen && j < ylen){
    if(x[i].index == y[j].index)
        sum += x[i++].value * y[j++].value;
    else {
        if(x[i].index > y[j].index)
            ++j;
        else
            ++i;
    }
}
return sum;
}

```

判别函数的向量化表示  $g(x)$  的形式是:  $g(x)=\text{sign}(\langle w, x \rangle + b)$ 。 $\langle w, x \rangle$  也记为  $w^T \cdot x$ , 也就是向量  $w$  的转置点乘向量  $x$ ,  $\text{sign}$  是符号函数。符号函数  $\text{sign}(a)$  的定义是: 当  $a > 0$  时, 则返回 1; 当  $a=0$  时, 则返回 0; 当  $a < 0$  时, 则返回 -1。有时候也把符号函数写作  $\sigma$ 。

一般把  $w$  称作权重向量 (weight vector), 把  $b$  叫作偏移量 (bias)。 $\langle w, x \rangle + b = 0$  所定义的面叫作超平面 (hyperplane)。

每一个训练样本由一个向量 (特征空间中的值组成的向量) 和一个标记 (标示出这个样本属于哪个类别) 组成, 记作  $D_i = (x_i, y_i)$ 。其中,  $y$  的取值只有两种可能: 1 和 -1 (分别用来表示属于  $C_+$  类, 还是属于  $C_-$  类)。

一般来说, 如果超平面远离分类训练集中的点应该会最小化对新数据错误分类的风险。点  $i$  到平面  $\Pi_{w,b}$  的距离  $d(\Pi_{w,b}, x_i) = |w \cdot x_i + b| / \|w\|$ 。 $d(\Pi_{w,b}, x_i)$  有时候也写作  $\delta_i$ 。

选取  $w$  和  $b$ , 使得超平面到最近点的距离最大。也就是求解  $\max_{(w,b)} (\min_i d(\Pi_{w,b}, x_i))$ , 这里的  $\|w\|$  叫作向量  $w$  的欧几里德范数 (Euclidean norm), 计算公式如下所示。

$$\|w\| = \sqrt{w_1^2 + w_2^2 + \dots + w_p^2}$$

对于距离超平面最近的  $C_+$  类的点  $x_+$  来说,  $w^T \cdot x_+ + b = 1$ ; 对于距离超平面最近的  $C_-$  类的点  $x_-$  来说,  $w^T \cdot x_- + b = -1$ 。



$$\max_{(w,b)} (\min_i d(\prod_{w,b}, x_i)) = \frac{|w \bullet x_- + b| + |w \bullet x_+ + b|}{\|w\|}$$

也就是求解基本的问题：在  $y_i(w^T x_i + b) \geq 1$  的条件下，求最小值  $\min_{w,b} \frac{1}{2} \|w\|^2$ 。

满足相等约束条件的这些点叫作支持向量（support vector），因为这些点的支持（约束）超平面。用拉格朗日乘子（Lagrange multiplier）来解决线性约束下的优化问题。

举个用拉格朗日乘子求解极值的例子。把一个拉格朗日乘子的函数整合进需要求最大或最小值的表达式。 $F(x,y)=x+2y$  有约束条件： $g(x,y)=x^2+y^2-4=0$ ，则引入一个拉格朗日乘子后，对应拉格朗日函数  $L(x,y,\lambda)=x+2y+\lambda(x^2+y^2-4)$ ，求导数。

$$\frac{\partial L}{\partial x} = 1 + 2\lambda x = 0 \quad (1)$$

$$\frac{\partial L}{\partial y} = 2 + 2\lambda y = 0 \quad (2)$$

$$\frac{\partial L}{\partial \lambda} = x^2 + y^2 - 4 = 0 \quad (3)$$

首先根据（1）、（2）、（3）解出拉格朗日乘子  $\lambda$  的值，然后计算出  $f(x,y)$  的最小值。对于线性可分的问题，可以通过二次规划（Quadratic programming）计算出拉格朗日乘子  $\lambda$  的值。

文本分类抽象成对空间中的点分类。举个对图 8-3 二维空间中的样本点分类的简单的例子。假设标志成+的点为： $\{(3,1),(3,-1),(6,1),(6,-1)\}$ ，标志成-的点为： $\{(1,0),(0,1),(0,-1),(-1,0)\}$ 。

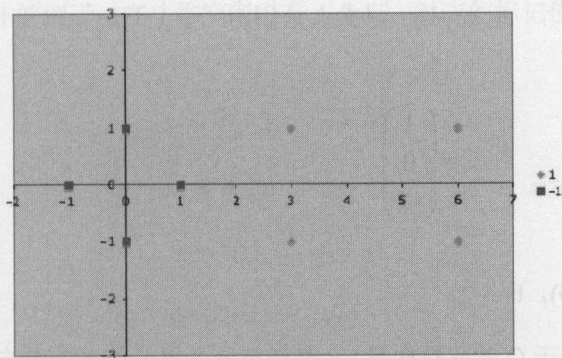


图 8-3 二维空间中的样本点

因为数据是线性可分的, 可以使用一个线性 SVM, 也就是说  $\Phi$  就是原函数。有 3 个支持向量:  $s_1=(1,0), s_2=(3,1), s_3=(3,-1)$ , 增加一个偏移量输入 1, 则  $\tilde{s}_1=(1,0,1), \tilde{s}_2=(3,1,1), \tilde{s}_3=(3,-1,1)$ 。

计算拉格朗日乘子  $\alpha$ 。

$$\alpha_1 * \tilde{s}_1 \bullet \tilde{s}_1 + \alpha_2 * \tilde{s}_2 \bullet \tilde{s}_1 + \alpha_3 * \tilde{s}_3 \bullet \tilde{s}_1 = -1$$

$$\alpha_1 * \tilde{s}_1 \bullet \tilde{s}_2 + \alpha_2 * \tilde{s}_2 \bullet \tilde{s}_2 + \alpha_3 * \tilde{s}_3 \bullet \tilde{s}_2 = +1$$

$$\alpha_1 * \tilde{s}_1 \bullet \tilde{s}_3 + \alpha_2 * \tilde{s}_2 \bullet \tilde{s}_3 + \alpha_3 * \tilde{s}_3 \bullet \tilde{s}_3 = +1$$

计算支持向量的点积, 得到结果如下所示。

$$2\alpha_1 + 4\alpha_2 + 4\alpha_3 = -1$$

$$4\alpha_1 + 11\alpha_2 + 9\alpha_3 = +1$$

$$4\alpha_1 + 9\alpha_2 + 11\alpha_3 = +1$$

求解得到:  $\alpha_1 = -3.5, \alpha_2 = 0.75, \alpha_3 = 0.75$ 。

$w$  可以表示成支持向量的线性组合:

$$\tilde{w} = \sum_i \alpha_i \tilde{s}_i = \alpha_1 * \tilde{s}_1 + \alpha_2 * \tilde{s}_2 + \alpha_3 * \tilde{s}_3$$

$$= -3.5 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + 0.75 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix}$$

求解结果是  $w=(1,0), b=-2$ 。

设  $x_+$  是任意一个属于  $C_+$  的支持向量,  $x_-$  是任意一个属于  $C_-$  的支持向量。而且, 总是至少会存在一个  $x_+$ , 也总是至少会存在一个  $x_-$ 。

根据等式:  $w^T x_+ + b = 1$  和  $w^T x_- + b = -1$ , 推导出  $b$  的另一个计算公式如下所示。

$$b = -\frac{1}{2}(w^T x_+ + w^T x_-)$$

$$b = -\frac{1}{2}(w^T s_2 + w^T s_1) = -\frac{1}{2}\left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}(3,1) + \begin{pmatrix} 1 \\ 0 \end{pmatrix}(1,0)\right) = -\frac{1}{2}(3+1) = -2$$

$W_2=0$  意味着特征空间的点的第二个维度对分类结果没有影响。判别条件是: 如果  $x_1 > 2$ , 则该点属于+类; 如果  $x_1 < 2$ , 则该点属于-类。分类超平面如图 8-4 所示。

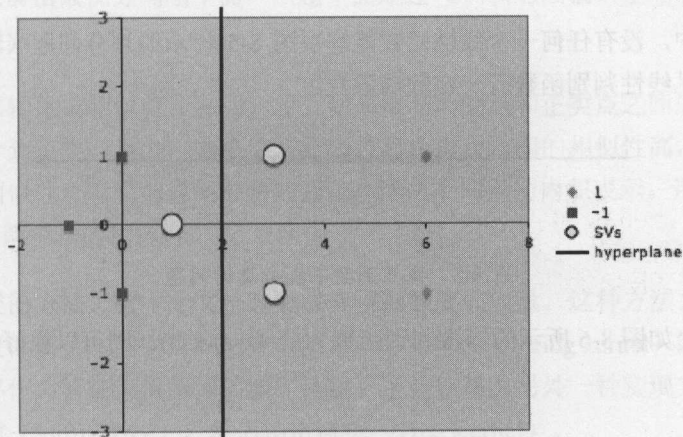


图 8-4 超平面

另外一个计算 SVM 的例子: 在二维空间中有 4 个点, 对应的标记值是  $y$ 。4 个点描述如下所示。

$$x_1 = (-2, -2) \quad y_1 = +1$$

$$x_2 = (-1, 1) \quad y_2 = +1$$

$$x_3 = (1, 1) \quad y_3 = -1$$

$$x_4 = (2, -2) \quad y_4 = -1$$

在  $\alpha \geq 0$  的条件下, 求下面这个拉格朗日函数的最大值。



$$L(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$= \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - 4\alpha_1^2 - \alpha_2^2 - \alpha_3^2 - 4\alpha_4^2 - 4\alpha_1\alpha_3 - 4\alpha_2\alpha_4$$

得到  $\alpha_1=0$ ,  $\alpha_2=\frac{1}{2}$ ,  $\alpha_3=\frac{1}{2}$ ,  $\alpha_4=0$ , 因此支持向量是  $x_2$  和  $x_3$ 。

$$w = \sum_i \alpha_i y_i x_i = \frac{1}{2}(x_2 - x_3) = (-1, 0)$$

$$b = y_2 - w^T x_2 = 0$$

在一维空间中, 没有任何一个线性函数能解决图 8-5 所示的划分问题 (粗线和细线各代表一类数据), 可见线性判别函数有一定的局限性。



图 8-5 线性函数不能分类的问题

如果建立一个如图 8-6 所示的二次判别函数  $g(x)=(x-a)(x-b)$ , 则可以很好地解决图 8-5 所示的分类问题。

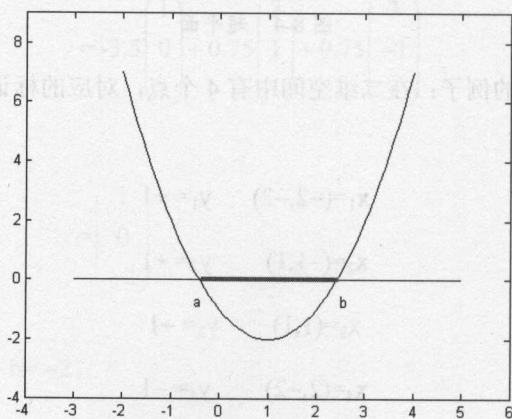


图 8-6 二次判别函数

这里  $g(x)=w_1*x^2+w_2*x+b$ 。以前的特征空间只有一个维度，这个维度对应的值也就是  $x$  的值。现在的特征空间有两个维度，第一个维度对应的值还是  $x$  的值，第二个维度对应的值是  $x^2$  的值。

举个日常生活中的例子：假设有个十字路口，南来北往的车辆连续不停会让这个十字路口拥堵，即使有红绿灯也无法完全解决问题，因为从平面交叉的二维角度来看才会无解。如果在这里建一个立交桥，车流就会变得顺畅无阻。也就是说加入“高度”这个维度，进入三维空间，问题就得到更好解决。因此，可以通过核函数把特征空间映射到更高的维度，这样有可能找到更好的超平面。图 8-6 是多项式核，此外常用的还有 RBF 核。

使用一个映射函数  $\Phi$  把输入空间中的数据转换成特征空间中的数据，使得分类问题变成线性可分的，然后求解出最优分隔超平面。当超平面通过  $\Phi^{-1}$  映射回输入空间时，超平面变成了一个复杂的决策表面。

分类问题可以转化成相似度计算的问题。如果待分类的点和正类点之间的相似性高，则可以把待分类的点分到正类；反之，如果待分类的点和负类点之间的相似性高，则可以把待分类的点分到负类。相似性可以用向量间夹角的余弦，或两个向量的内积表示。用核函数  $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$  来度量两个点的相似性。

回顾文本分类的方法，把一个文档映射成有很高维度的向量。这种方法丢失了所有的词的顺序信息，而仅仅保留了文档中的词的频率信息。字符串核（String kernel）通过从文档中提取  $k$  个前后相连的字作为特征来保留词的顺序信息。字符串核的另外一种实现方法是：首先把字符串转换成后缀树（Suffix Tree），然后构造树核（Tree Kernel）。

因为训练集中噪声的存在，尤其是训练集不能代表全部判决空间，所以正确处理相对模糊的间隔区域（margin）与全集的比例是关键。最优分类超平面的定义是指：该分类面不但能正确分类，而且能使各类别的分类间隔最大。最优训练的含义是：在确定的平面分类间隔条件下，不但训练序列的判决正确率高，而且推广到全集序列的判决率也尽可能高。

SVM 最基本的方法只能实现两类分类，可以通过组合多个两类分类器来实现多类分类。例如，有  $N$  类，一种解决方法是学习  $N$  个 SVM 分类器。

```
SVM 1 学习 "Output==1" vs "Output != 1"
SVM 2 学习 "Output==2" vs "Output != 2"
.....
SVM N 学习 "Output==N" vs "Output != N"
```

计算文档属于每个类别的隶属度，然后取最大隶属度对应的类别。这个方法叫作一类对余类，取最大隶属度对应的类别实现代码如下所示。

```
//输入每个类别的隶属度组成的数组
//得到文档的所属类别 ID
private int singleCategory(double[] simRatio) {
    int catID = 0;
    double maxNum = simRatio[catID];
    for (int i = 1; i < m_nClassNum; i++) {
        if (simRatio[i] > maxNum) {
            maxNum = simRatio[i];
            catID = i;
        }
    }
    return catID;
}
```

另外介绍一种两类分类器来实现多类分类的方法，叫作错误校正输出编码（Error Correcting Output Coding）。例如，有  $m = 4$  个类别，分别是：政治、体育、商业、艺术。

分配唯一的  $n$  位向量给每个类名， $n > \log_2 m$ 。第  $i$  个位向量作为类名  $i$  的唯一的编码。 $m$  个类名组成的位矩阵记作  $C$ ，如表 8-5 所示。

表 8-5 类名编码

类 名	编 码
政治	0110110001
体育	0001111100
商业	1010101101
艺术	1000011010

对每列构建独立的二元分类器。这里是 10 个分类器。正类文本是  $C_{ij} = 1$  对应的类别，负类文本是  $C_{ij} = 0$  对应的类别。例如，第三个分类器把{体育、艺术}作为负类，{政治、商业}作为正类。

通过插件分类器判断文档类别。把预测某一位的值的分类器叫作插件分类器，一个插件分类器预测文档属于某个类别的子集，根据  $\{\lambda^1, \lambda^2, \dots, \lambda^n\}$  来判断文档的类别。

计算文档  $x$  的类别时，先生成一个  $n$  位的向量  $\lambda(x) = \{\lambda_1(x), \lambda_2(x), \dots, \lambda_n(x)\}$ 。



生成的位向量  $\lambda(x)$  很有可能不是  $C$  中的一行, 但是可能更像某些行, 也就是和某些行的海明距离更近。把文档分类成这行对应的类别  $\operatorname{argmin}_i \operatorname{HamingDistance}(C_i, \lambda(x))$ , 可以根据海明距离判断  $\lambda(x)$  和哪行最相似。

假设  $C_i$  和  $\lambda(x)$  最相似, 则第  $i$  个类别作为文档  $x$  的类别。如果生成的位向量是  $\lambda(x) = \{1010111101\}$ , 则把这篇文档分到商业类。

自动分类的 SVM 方法接口分为训练过程接口和执行分类的接口。分类器训练过程的接口如下所示。

```
//创建一个分类器
Classifier svmClassifier = new Classifier();
//设置训练文本的路径
svmClassifier.setTrainSet("D:/train");
//设置训练输出模型的路径
svmClassifier.setModel("D:/model");
//执行训练
svmClassifier.train();
```

把分类器训练好的结果写入  $D:/model$  目录, 然后执行分类的接口如下所示。

```
//加载已经训练好的分类模型
Classifier theClassifier = new Classifier("D:/model/model.prj");
//文本分类的内容
String content = "我要买把吉他, 希望是二手的, 价格 2000 元以下";
//开始使用 SVM 方法分类
String catName = theClassifier.getCategoryName(content);
System.out.println("类别名称:" + catName);
```

## 8.2.4 多级分类

以二级分类为例, 在路径  $D:\text{train}\backslash\text{zippo}$  收藏乐器, 建立子路径如下所示。

```
D:\train\zippo 收藏乐器\打火机 zippo 烟具
D:\train\zippo 收藏乐器\古董收藏
D:\train\zippo 收藏乐器\乐器乐谱
D:\train\zippo 收藏乐器\邮币卡字画
```

训练二级分类的程序如下所示。

```
//训练主分类
String strPath = "D:/lg/work/xiaoxishu/train";
String modelPath = "D:/lg/work/xiaoxishu/model";
Classifier svmClassifier = new Classifier();
svmClassifier.setTrainSet(strPath);
svmClassifier.setModel(modelPath);
svmClassifier.train();
File dir = new File(strPath);
File[] files = dir.listFiles();
for (int i = 0; i < files.length; i++){
    File f = files[i];
    if (f.isDirectory()) {
        //训练子分类
        System.out.println(f.getAbsolutePath().getName());
        String subTrain = strPath + "/" + f.getAbsolutePath().getName();
        Classifier subClassifier = new Classifier();
        subClassifier.setTrainSet(subTrain);
        String subModelPath = modelPath + "/" + f.getAbsolutePath().getName();
        subClassifier.setModel(subModelPath);
        subClassifier.train();
    }
}
```

分类的执行过程如下所示。

```
String modelPath = "D:/lg/work/xiaoxishu/model";
Classifier theClassifier = new Classifier(modelPath+"/model.prj");
String content = "我要买把吉他，希望是二手的，价格 2000 元以下"; //分类文本内容
System.out.println("分类开始");
String catName = theClassifier.getCategoryName(content);
System.out.println("catName:"+catName);
if (catName == null){
    //如果没有主分类则返回
    return;
}
String subModelPath = modelPath+"/"+catName+"/model.prj";
Classifier subClassifier = new Classifier(subModelPath);
String subCatName = subClassifier.getCategoryName(content);
System.out.println("subCatName:"+subCatName);
```

上面的执行结果将打印出。

```
//分类开始
catName:zippo 收藏乐器
subCatName:乐器乐谱
```

也就是把“我要买把吉他，希望是二手的，价格 2000 元以下”分成大类属于“zippo 收藏乐器”，子类是“乐器乐谱”。

## 8.2.5 网页分类

抓取的新闻网站，例如 <http://news.sina.com.cn/> 网站上面本来就有国际新闻和国内新闻的栏目。如果索引页能分成“国际新闻”或“国内新闻”中的一种，详细页的类别参考索引页的分类，那么就可以形成一个“国际新闻”和“国内新闻”的分类训练样本库。

此外，从 <http://bj.58.com/job.shtml> 网址就可以知道它可能是和招聘相关的网页，而且可能是北京地区的网页。因此，使用 URL 地址可以快速分类网页。例如，凤凰网中的新闻的 URL 地址是 [http://news.ifeng.com/mil/4/detail\\_2010\\_09/11/2489355\\_0.shtml](http://news.ifeng.com/mil/4/detail_2010_09/11/2489355_0.shtml)，URL 中包含 reading 的就分为读书类，包含 mil 的就分为军事类，包含 culture 的就分为文化类。

为了提取“<http://bj.58.com/job.shtml>”的特征词“bj”“58”“job”，首先按“:”“/”“.”切分，然后通过词干化和小写化处理，去除停用词“http”“com”“shtml”，最后剩下“bj”“58”“job”3 个分类特征词。

有些 URL 存在共同的前缀，这样的是在一个模块的，但是有些是二级域名，还有些比较乱。复杂首页很难自动理顺，也许网站自己的有点乱，因为是从列表页采集的。看能否根据同一个目录下的多个详细页来自动分类，共同决定这个目录页的类别。

## 8.3 本章小结

本章介绍了一些经典的文本分类和网页分类方法。深度学习中有的一些简单而有效的文本分类方法，其中的 fastText 文本分类方法，可以在普通 CPU 上快速训练模型。



# 9

## 第9章

---

### 案例分析

信息采集往往是大的软件系统中的一部分，本章介绍爬虫在软件系统中的应用。

#### 9.1 金融爬虫

有各种可投资的产品，这里介绍从能源期货到股票数据分析。

##### 9.1.1 中国能源政策数据

从中国能源网站、发改委能源局网站、国家统计局网站等网，抓取 2000 年至 2016 年中国能源政策数据，例如，<http://www.cnenergy.org/>。

## 9.1.2 世界原油现货交易和期货交易数据

抓取 2010 年至 2016 年世界原油现货交易和期货交易数据。国外数据网站, 例如 IEA (<http://www.iea.org/>)、BP 等。

找到要抓取的期货的代号, 例如, NYMEX 原油的代号是 CL, 它的接口就是 [http://hq.sinajs.cn/list=hf\\_CL](http://hq.sinajs.cn/list=hf_CL), 只要访问这个地址, 就会发现如下所示一串字符串, 该字符串代表了访问时这期期货的实时数据。

```
var
hq_str_hf_CL="82.90,0.1813,82.89,82.90,83.48,82.67,19:20:37,82.75,83.09,804,0,0,2014-10-20,NYMEX 原油";
```

这串数据包含了期货的最新价格、昨收、今开、最高价、最低价等信息。

## 9.1.3 股票数据

公司基本信息的提取: 上市公司总部办公地点的空气质量可能对股价有影响, 公司办公地址可以看成是短文本。例如, 广东省深圳市南山区西丽松白路 1061 号茂硕科技园。

用几千个的气象台监测地名匹配公司的办公地址。

```
//从前往后匹配, 得到在地址中最开始出现的气象台地名
for(int i=0;i<address.length();++i){
    String stationCode = dic.matchLong(address, i);
    if(stationCode!=null){ //找到气象台地名对应的城市编码
        return Aqi.getAQByCode(stationCode); //得到天气质量
    }
}
```

抓取证券交易所上市公司的公告。A 股有两个交易所, 分别是深交所和上交所。深交所的公告参见 <http://disclosure.szse.cn/m/drgg.htm>, 上交所的公告参见 [http://www.sse.com.cn/disclosure/listedinfo/announcement/s\\_docdatesort\\_desc.htm](http://www.sse.com.cn/disclosure/listedinfo/announcement/s_docdatesort_desc.htm)。深交所网站有时候不太稳定, 巨潮资讯网可以看作是它的镜像。对每条公告按重要度打分, 把公告条目按重要度排序后输出。可以根据这些 PDF 文件填充知识库, 知识库的一种表示形式: 实体属性。例如, 运达科技, 营业额, 12 亿。

根据上下文提取数字, 例如, 从公告 PDF 文件中提取股东人数。文档格式存在好几种写法,

例如，股东总数（户）35 482，报告期末普通股东总数 6 574，所以定义几个规则来对应不同的写法。

```
RuleParser rp = new FSTParser(); //规则解析器
TextExtractor ie = new TextExtractor(rp); //文本提取器

String right = "股东总数（户） <num>{holdernum}";
ie.add(right);

right = "股东总数 <num>{holdernum}";
ie.add(right);
```

提取代码如下所示。

```
String text = "报告期末普通股股东总数 6 574 "; //待提取的文本
AdjList g = ie.getLattice(text); //形成词图
String source = g.findType("holdernum"); //发现股东人数
System.out.println("提取股东人数:" + source); //从文本中提取股东人数
```

识别数字的文法（BNF）。

```
floatnumber ::= pointfloat | exponentfloat
pointfloat ::= [intpart] fraction | intpart "."
exponentfloat ::= (intpart | pointfloat) exponent
intpart ::= digit+
fraction ::= "." digit+
exponent ::= ("e" | "E") ["+" | "-"] digit+
```

自动机和文法可以等价转换，识别数字的自动机代码如下所示。

```
Automaton a = BasicAutomata.makeCharRange('0', '9');
Automaton b = a.repeat(1); //出现至少一次
Automaton comma = BasicAutomata.makeChar(',');
Automaton end = BasicOperations.concatenate(comma, a.repeat(1)); //串联
Automaton intNum = BasicOperations.concatenate(b, end.repeat()); //整数部分
Automaton comma2 = BasicAutomata.makeChar('.'); //小数点
Automaton floatNum = BasicOperations.concatenate(comma2, a.repeat(1)); //浮点数部分
Automaton intWithFloat = BasicOperations.concatenate(intNum,
    floatNum.optional()); //浮点数部分是可选的
Automaton num = BasicOperations.union(intWithFloat, floatNum); //并联
num.determinize(); //NFA 转换成 DFA
```



对于数值类型非结构化信息的处理流程如下所示。

- (1) 从文本中提取特定含义的数值。
- (2) 归一化提取出来表示数量的文本字符串，并转换成数值。
- (3) 把提取出来的数值存入数据库。

-123 123.55 这样的数值归一化是信息提取中的一个问题。开源软件 `stanford.nlp.ie` 中有现成的实现方法。

从文本中提取特征。例如，判断某个上市公司是否是家族企业。

```
PersonTemplate template = new PersonTemplate();

//从数据库加载股票名称、人名特征词
Connection con = DBUtil.getConnect();
addNr(con,template); //人名特征词
addStock(con,template); //股票名

String pattern = "<StockName>家族企业";
template.addRule(pattern);

pattern = "<StockName>是个家族企业";
template.addRule(pattern);

//定义人名
String ruleName = "PersonName";
pattern = "<surName><singleName>"; //例如，王石
template.addRule(ruleName, pattern);

ruleName = "PersonName";
pattern = "<surName><doubleName1><doubleName2>"; //例如，郭广昌
template.addRule(ruleName, pattern);

//增加模板
pattern = "<StockName>实际控制人<PersonName>及<PersonName>夫妇";
template.addRule(pattern);
```

根据文本判断是否家族企业的例子。

```
PersonTemplate template = TemplateFactory.create();
```

```
String sentence = "王氏家族拥有吉祥航空";
boolean isFamilyHold = template.match(sentence);
System.out.println("是否家族企业 "+isFamilyHold);
```

### 9.1.4 从 PDF 文件中提取表格

tabula-java 使用扩展表格提取算法, 实现从 PDF 文件中提取表格, 该算法使用几何线条来重建表结构。丢弃斜线后, 发现所有线的交叉点, 使用这些交叉点创建一个很大的最小矩形区域列表。

先测试下 pdfbox 能否正确处理 PDF 文件。

```
java -jar pdfbox-app-2.0.0-RC1.jar PDFToImage f:/stock/百利电气半年报.pdf
```

线裁减算法: Cohen Sutherland line clipping algorithm, 测试代码如下所示。

```
Rectangle2D clip = new Rectangle2D.Float(); //矩形
clip setFrame(100, 150, 200, 100);
Line2D.Float clipee = new Line2D.Float(600f, 600f, 800f, 900f); //线
CohenSutherlandClipping clipping = new CohenSutherlandClipping(clip);
boolean clipped = clipping.clip(clipee); //裁减
System.out.println(clipped); //看线是否被裁减过
```

## 9.2 商品搜索

官方商城购物搜索, 爬虫抓取所有的官方商城, 例如, 阿迪达斯、小米等官方网站。

上网买东西, 如果要去每家网站找东西并比较工作量很大, 那么某个网站自动分析, 呈现出来, 还能及时享受今天各大商场的优惠就很轻松了。

商品爬虫就抓那几百个以上的商品网站, 需要手机客户端找到商品, 商品评价、相关推荐等。

抓取 <http://www.xiu.com/> 中的商品, 然后导入自己的网上商城。网上商城是用 ECShop 软件搭建的。采用 HttpClient 下载网页, Jsoup 分析页面。根据输入 URL 返回 InputSource。

```

public synchronized static InputSource getInputSource(String url) throws
ClientProtocolException, IOException{
    HttpClient httpclient = HttpClientConnManager.getMultiThreadedClient();
    HttpGet httpget = new HttpGet(url);

    HttpEntity entity = httpclient.execute(httpget).getEntity();

    InputSource is = new InputSource(entity.getContent());
    is.setEncoding("utf-8");
    return is;
}

```

是否有新的商品，如果没有新商品，则停止翻页。

商品网站的列表页是 <http://list.xiu.com/19380.html>，详细页是 <http://item.xiu.com/product/0359097.shtml>。首先实现提取详细页的信息，然后遍历列表页。

提取标题。

```

<div class="p_title">
    <span>【广货网上行】 ZIMMUR 桑蚕丝 欧美风 新中袖真丝连衣裙 (DV-0383) </span>
    <span class="cp"><font>品 牌: </font><a
href="http://brand.xiu.com/2439.html">ZIMMUR</a></span>
</div>

```

提取商品信息。

```

Elements links = doc.select("div.p_title");
String name = links.get(0).childNodes(1).childNodes(0).toString();
//提取商品信息

```

提取内容。

```

<div class="conlist">
ZIMMUR 服装品牌永远相信镜子中旋转的你，这和任何人无关，它会告诉你春天的原野，夏天的海风，秋天的红叶，冬天的
恋歌。它是徜徉在人生韶华中信步而来的风景，它是款款而来的你，它是延时的青春。自由来自于浪漫，浪漫永恒于经典，
经典铭刻于时尚。ZIMMUR 给你史诗般的感觉。
</div>

```

根据 class 类别提取信息。

```

Element link = doc.select("div.conlist").get(2);
String desc = link.text();

```



提取图片信息。

```

```

提取图片代码，也就是 id，它是 imgPic 的标签。

```
Element content = document.select("#imgPic").first();
```

提取缩略图。

```
  
document.select("img[onload]"); //包含 onload 属性的 img 标签
```

处理翻页参数 <http://list.xiu.com/20069.html?currentPage=2>，去掉无关的参数。

```
public static String getListURL(String oldURL){  
    ParseURL splitURL = new ParseURL(oldURL);  
    if(splitURL.searchparms==null){  
        return splitURL.baseURL;  
    }  
    String curPage = splitURL.searchparms.get("currentPage");  
    if (curPage != null)  
        return splitURL.baseURL + "?currentPage=" + curPage;  
    return splitURL.baseURL;  
}
```

## 9.2.1 遍历商品

如何得到所有商品的详细信息？<http://www.xiu.com/about/classifymap.shtml> 包含了所有的商品类别，把这个问题分解为抓取所有类别的商品。

把每个类别下的商品进一步分解为类别首页所能访问到的商品，其中包含类别首页的商品，或者它直接指向或者间接指向的目录页。图 9-1 中的小球表示商品详细页，每个目录页都包含

一些详细页面。

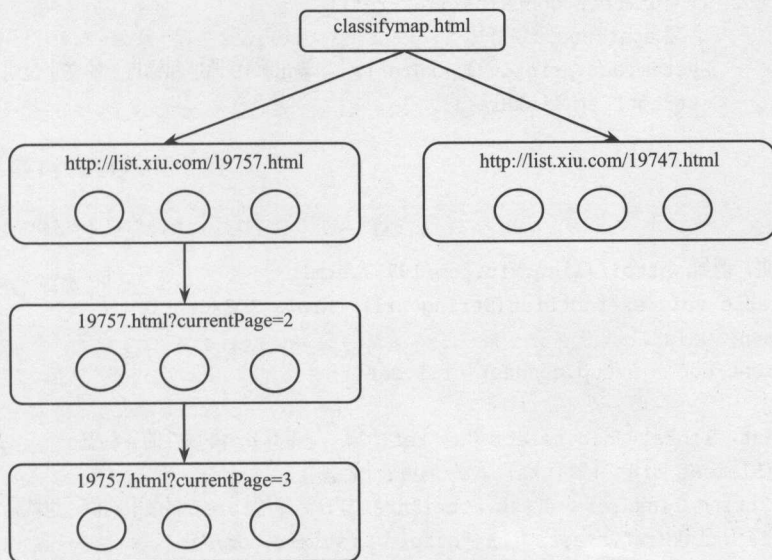


图 9-1 商品网站有效链接之间的关系

用分而治之的方法实现遍历所有商品的问题，往往采用递归调用解决问题的方法来实现分而治之。一个目录页下面的所有商品，包括它们之间链接的商品，以及它指向的目录页链接的商品。一个目录页下面的所有商品就是一个可以再次分解的问题，采用递归调用的方式遍历目录页。

`extractList` 方法处理目录页，例如，`http://list.xiu.com/19757.html`。`getProduct` 方法处理详细页，例如，`http://item.xiu.com/product/0359097.shtml`。完整的过程如下所示。

```

public class ExtractProduct {
    //优化成包含 2 万个网址
    static BloomFilter<String> urlSeen = new BloomFilter<String>(200000, 20000);

    public static void main(String[] args) throws IOException {
        String url = "http://www.xiu.com/about/classifymap.shtml";
        Document doc = Jsoup.connect(url).get();
        Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
        for (Element link : links) { //遍历每个链接

```

```

        String linkHref = link.attr("href"); //得到 href 属性中的值, 也就是 URL 地址
        if (linkHref.startsWith("http://list.xiu.com/")
            || linkHref.startsWith("http://brand.xiu.com/")) {
            if (urlSeen.contains(linkHref))
                continue;
            System.out.println(linkHref); //输出 URL 地址和锚点上的文字说明
            extractList(linkHref);
        }
    }
}

//处理目录页, 例如, http://list.xiu.com/19757.html
public static void extractList(String url) throws IOException {
    urlSeen.add(url);
    Document doc = Jsoup.connect(url).get();

    Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
    for (Element link : links) { //遍历每个链接
        String linkHref = link.attr("href"); //得到 href 属性中的值, 也就是 URL 地址
        if (linkHref.startsWith("http://list.xiu.com/")
            || linkHref.startsWith("http://brand.xiu.com/")) {
            if (urlSeen.contains(linkHref))
                continue;
            extractList(linkHref); //递归调用
        } else if (linkHref.startsWith("http://item.xiu.com/product/")) {
            if (urlSeen.contains(linkHref))
                continue;
            getProduct(linkHref); //提取详细页中的商品信息
        }
    }
}

//处理详细页
public static void getProduct(String url) throws IOException {
    urlSeen.add(url);
    Document doc = Jsoup.connect(url).get();
    Elements links = doc.select("div.p_title");
    String name = links.get(0).childNodes(1).childNodes(0).toString(); //提取名称

    Element link = doc.select("div.conlist").get(2);
    String desc = link.text(); //提取商品描述
}

```



```

Element img = doc.select("#imgPic").first();
String imgSrc = img.attr("src"); //提取商品图片

insertDatabase(url,name,desc,imgSrc); //把提取的商品信息放入数据库
}
}

```

写入商品数据的 SQL 语句。

```
insert into good(url,name,desc,img) values(?,?,?,?)
```

使用 JDBC 写入商品。

```

public static void insertDatabase(String url, String name, String desc,
    String img) {
    Connection conn = null;
    PreparedStatement stmt = null;
    PreparedStatement stmt2 = null;
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        conn = DriverManager
            .getConnection("jdbc:odbc:driver={Microsoft Access Driver
(*.mdb)};DBQ=good.mdb");

        stmt2 = conn
            .prepareStatement("insert into good (url,name,desc,img)
values(?,?,?,?)");
        stmt2.setString(1, url);
        stmt2.setString(2, name);
        stmt2.setString(3, desc);
        stmt2.setString(4, img);
        stmt2.executeUpdate();
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        try {
            if (stmt != null)
                stmt.close();
            if (stmt2 != null)
                stmt2.close();
            if (conn != null)

```

```

        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

除了采用递归调用的方式，还可以把列表页放入队列。

```

static Deque<String> listQueue = new ArrayDeque<String>(); //保存待处理的目录页链接

public static void main(String[] args) throws IOException {
    String url = "http://www.xiu.com/about/classifymap.shtml";
    Document doc = Jsoup.connect(url).get();
    Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
    for (Element link : links) { //遍历每个链接
        String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
        if (linkHref.startsWith("http://list.xiu.com/"))
            || linkHref.startsWith("http://brand.xiu.com/")) {
            if (urlSeen.contains(linkHref))
                continue;
            listQueue.addLast(linkHref); //加入初始目录页
        }
    }

    //遍历每个目录页
    while(!listQueue.isEmpty()){
        String listURL = listQueue.pollFirst(); //从队列中取得一个目录页
        extractList(listURL); //处理这个目录页
    }
}

public static void extractList(String url) {
    urlSeen.add(url);
    Document doc = Jsoup.connect(url).get();

    Elements links = doc.select("a[href]"); //带有 href 属性的 a 标签
    for (Element link : links) { //遍历每个链接
        String linkHref = link.attr("href"); //得到 href 属性中的值，也就是 URL 地址
        if (linkHref.startsWith("http://list.xiu.com/"))
            || linkHref.startsWith("http://brand.xiu.com/")) { //目录页
            if (urlSeen.contains(linkHref))

```

```

        continue;
        listQueue.addLast(linkHref);
    } else if (linkHref.startsWith("http://item.xiu.com/product/")) { //详细页
        if (urlSeen.contains(linkHref))
            continue;
        getProduct(linkHref);
    }
}
}
}

```

## 9.2.2 使用 HttpClient

Jsoup 下载网页的方法太简单, 所以用 HttpClient 下载网页, 然后把网页内容交给 Jsoup 解析。使用 HttpClient 的项目中需要用到 httpcore-4.2.1.jar、httpclient-4.2.1.jar 和 commons-logging-1.1.1.jar 三个 jar 包。可以把与日志相关的 jar 文件和配置文件都放入 lib 目录, 并把 lib 目录设置成源代码目录。

```

public static String httpClientGet(String url){
    StringBuilder pageBuffer = new StringBuilder();
    //创建一个客户端, 类似于打开一个浏览器
    DefaultHttpClient httpClient = new DefaultHttpClient();

    //创建一个 GET 方法, 类似于在浏览器地址栏中输入一个地址
    HttpGet httpget = new HttpGet(url);

    //类似于在浏览器地址栏中输入回车, 获得网页内容
    HttpResponse response = httpClient.execute(httpget);

    //查看返回的内容, 类似于在浏览器查看网页源代码
    HttpEntity entity = response.getEntity();
    if (entity != null) {
        //读入内容流, 并以字符串形式返回, 这里指定网页编码是 UTF-8
        InputStream stream = entity.getContent();

        //缓存读入数据
        Scanner scanner = new Scanner(
            new InputStreamReader(stream, "utf-8"));
        scanner.useDelimiter("\\\\z");
        //读取网页内容
    }
}

```



```

        while (scanner.hasNext()) {
            pageBuffer.append(scanner.next());
        }
        EntityUtils.consume(entity); //关闭内容流
    }
    //释放连接
    httpClient.getConnectionManager().shutdown();

    return pageBuffer.toString();
}

```

把 HttpClient 下载的网页内容交给 Jsoup 解析。

```

String content = httpClientGet(url); //得到网页内容
Document doc = Jsoup.parse(content);

```

### 9.2.3 提取价格

查看 <http://item.xiu.com/product/0359097.shtml> 网页的源代码，看不到商品的价格，只有在浏览器中选择对应的元素后用 Firebug 打开，才能看到价格对应的网页标签。

用户加载网页，加载静态页面结束后，浏览器会执行网页中的初始化 JavaScript。初始化 JavaScript 请求后台的服务器端 URL。服务器端发送商品价格等数据给 JavaScript。JavaScript 修改静态页面，增加价格信息，得到最终用户看到的界面。

用 Firebug 跟踪所有的请求，检查其中的 HTML 类型的请求。发现是通过 [http://portal.xiu.com/business/GoodsDetailMainCmd?langId=-7&storeId=10154&catalogId=10101&jsoncallback=jQuery16206338461217635925\\_1346634902425&rand=&goodsId=2325328&goodsSn=30004874&sellType=5&\\_1346634902731](http://portal.xiu.com/business/GoodsDetailMainCmd?langId=-7&storeId=10154&catalogId=10101&jsoncallback=jQuery16206338461217635925_1346634902425&rand=&goodsId=2325328&goodsSn=30004874&sellType=5&_1346634902731) 网址请求的价格。

返回 JSON 格式的数据。

```

jQuery16209120799453623849_1346559933535([{"totalLimitNumber":0,"onSaleType":0,"saleType":
":5","priceType":"","nowTime":"2012-09-02
12:25:50","onSale":1,"installmentInfo":null,"endTime":"","preSaleGoods":null,"activity":
null,"colors":[{"attrValue":"黑色
","attrName":"","attrId":"70000000000000000133"}, {"attrValue":"杏色
","attrName":"","attrId":"70000000000000000158"}], "sizes":[{"attrValue":"S", "attrName":"","
","attrId":"700000000000000009192"}, {"attrValue":"M", "attrName":"","attrId":"70000000000000

```

```

09056"}, {"attrValue": "L", "attrName": "", "attrId": "7000000000000009191"}, {"attrValue": "XL",
, "attrName": "", "attrId": "7000000000000009095"}], "offSaleShow": 0, "sizeColorStock": [{"size
Value": "S", "stock": 0, "skuId": "359098", "skuCode": "110258650001", "sizeId": "700000000000000
9192", "colorValue": "黑色
", "colorId": "7000000000000000133"}, {"sizeValue": "M", "stock": 0, "skuId": "359099", "skuCode"
: "110258650002", "sizeId": "7000000000000009056", "colorValue": "黑色
", "colorId": "7000000000000000133"}, {"sizeValue": "L", "stock": 0, "skuId": "359100", "skuCode"
: "110258650003", "sizeId": "7000000000000009191", "colorValue": "黑色
", "colorId": "7000000000000000133"}, {"sizeValue": "XL", "stock": 0, "skuId": "359101", "skuCode
": "110258650004", "sizeId": "7000000000000009095", "colorValue": "黑色
", "colorId": "7000000000000000133"}, {"sizeValue": "S", "stock": 1, "skuId": "359102", "skuCode"
: "110258650005", "sizeId": "7000000000000009192", "colorValue": "杏色
", "colorId": "7000000000000000158"}, {"sizeValue": "M", "stock": 1, "skuId": "359103", "skuCode"
: "110258650006", "sizeId": "7000000000000009056", "colorValue": "杏色
", "colorId": "7000000000000000158"}, {"sizeValue": "L", "stock": 0, "skuId": "359104", "skuCode"
: "110258650007", "sizeId": "7000000000000009191", "colorValue": "杏色
", "colorId": "7000000000000000158"}, {"sizeValue": "XL", "stock": 1, "skuId": "359105", "skuCode
": "110258650008", "sizeId": "7000000000000009095", "colorValue": "杏色
", "colorId": "7000000000000000158"}], "stock": 1, "goods": {"goodsName": "ZIMMUR 桑蚕丝 欧美风 新
中袖真丝连衣裙
(DV-0383)", "catName": "", "PScore": "299", "fullCatId": "", "goodsCode": "", "catTopName": "", "in
vType": "1", "globalFlag": "0", "goodsSn": "11025865", "XPrice": "299.00", "catId": "", "goodsId":
"359097", "spaceFlag": "", "SPrice": "", "PPrice": "", "catTopId": "", "MPrice": "2390.00"}, "prefe
rential": {"detail": [], "activityPriceType": "0", "xiuPrice": 29900, "endTime": "", "discountPri
ce": 29900, "discountAmount": "0.00", "activityPrice": -100, "largess": []}, "colorsName": "颜色
", "limitGoods": {"startDate": "", "limitNumber": 10, "endDate": ""}, "sizesName": "尺码
", "combinationInfo": null, "errorMessage": ""}}))

```

其中包含了 AJAX 填充的价格, 如图 9-2 所示。

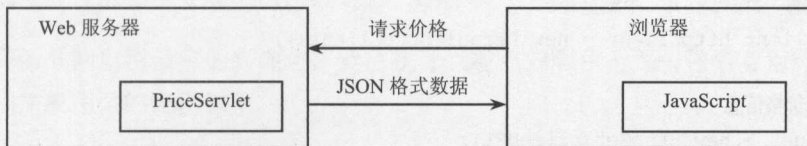


图 9-2 AJAX 填充的价格

使用简单的字符串查找方法提取价格。

```

String prefix = "\"xiuPrice\"";
int start = content.indexOf(prefix) + prefix.length();

```

```
int end = content.indexOf(start, ","); //找指定位置开始的逗号
String priceValue = content.substring(start,end);
```

也可以解析 JSON 格式的数据,然后提取想要的信息。可以使用 Jackson (<http://jackson.codehaus.org/>) 解析 json。Jackson 是一个高性能的 json 解析器。ObjectMapper.readValue 方法把字符串转换成 Java 类。

```
ObjectMapper mapper = new ObjectMapper(); //可以重用 ObjectMapper 对象
Map<String, List<String>> map = mapper.readValue (jsonStr, Map.class);
```

JSON-java 的介绍网站是 <http://www.json.org/java/index.html>。可以从 <https://github.com/douglascrockford/JSON-java> 上下载解析 JSON 格式数据的几个相关类。

```
String the_json="{ 'profiles': [{ 'name': 'john', 'age': 44}, { 'name': 'Alex', 'age': 11}]}";
```

返回 the\_json 字符串中的 profiles 数组。

```
JSONObject myjson = new JSONObject(the_json);
JSONArray the_json_array = myjson.getJSONArray("profiles");
```

得到价格信息。

```
String storeId_param = splitURL.searchparms.get("storeId");
String param_goodsId = param.attr("value");
String param_goodsSn = param.attr("value");

String priceURL = "http://portal.xiu.com/business/GoodsDetailMainCmd?storeId="
    + storeId_param
    + "&jsoncallback=jQuery162011208900439210956_1346568313493&goodsId="
    + param_goodsId + "&goodsSn=" + param_goodsSn;
```

//创建一个客户端,类似于打开一个浏览器

```
DefaultHttpClient httpClient = new DefaultHttpClient();
```

//GET 方法请求价格信息

```
HttpGet httpget = new HttpGet(priceURL);
```

//类似于在浏览器地址栏中输入回车,获得网页内容

```
HttpResponse response = httpClient.execute(httpget);
```

//查看返回的内容,类似于在浏览器中查看网页源代码

```
HttpEntity entity = response.getEntity();
```



```
//读入内容流,并以字符串形式返回,这里指定网页编码是 UTF-8
System.out.println(EntityUtils.toString(entity,"utf-8"));
EntityUtils.consume(entity);//关闭内容流

//释放连接
httpClient.getConnectionManager().shutdown();
```

使用 `JSONObject` 提取价格信息。`JSONTokener` 从字符串中解析出 `JSONObject` 对象,然后得到 "goods" 对象,最后得到其中 `Double` 类型的价格。

```
JSONTokener jsonParser = new JSONTokener(json_str);
//读取一个 JSONObject 对象
JSONObject person = (JSONObject) jsonParser.nextValue();
//接下来的就是 JSON 对象的操作
double local_price = person.getJSONObject("goods").getDouble("XPrice");
double market_price = person.getJSONObject("goods").getDouble("MPrice");
```

抓取两个不同页面的内容,整合到同一行。用 `update` 更新这行,补充数据。

## 9.2.4 水印

下载的商品图像中有些包括水印,需要判断一个图像是否存在水印。

图片中的水印添加:首先,输入图像。然后,指定水印位置,以及水印透明度。最后,输出图像。

感兴趣区域 (ROI) 是从图像中选择一个图像区域,这个区域是图像分析关注的重点。圈定该区域以便进行进一步处理,水印点是感兴趣区域。

用掩模图像分割出图像中的兴趣点,然后统计兴趣点的颜色。每个像素点对应掩模图像中的一位。得到掩模图像的流程如下。

- 去噪音:去掉个别孤立点。
- 灰度化:变成灰度图。
- 二值化:可以使用 OTSU 算法转换成黑白点组成的图。黑色的点是兴趣点,而白色的点则是背景。

图像有  $500 \times 500 = 250\,000$  个点,其中有 4 143 个兴趣点。

在兴趣点上，比较标准图像和检测图像，看检测像素是否加过水印。

```
int r1 = (detect[i] >> 16) & 0xff; //高8位
int g1 = (detect[i] >> 8) & 0xff; //中间8位
int b1 = (detect[i]) & 0xff; //低8位

int r2 = (stand[i] >> 16) & 0xff; //高8位
int g2 = (stand[i] >> 8) & 0xff; //中间8位
int b2 = (stand[i]) & 0xff; //低8位

//加上水印后，每种颜色的色彩饱和度都会降低
if( (r2 >= r1) && (g2 >= g1) && (b2 >= b1) )
    count++;
```

计算检测图片中符合条件的像素比例。如果这个值大于 0.75，就算有水印。

```
private static double sim(int[] detect, int[] stand) {
    int count=0;
    for (int i = 0; i < detect.length; ++i) {
        int r1 = (detect[i] >> 16) & 0xff; //高8位
        int g1 = (detect[i] >> 8) & 0xff; //中间8位
        int b1 = (detect[i]) & 0xff; //低8位

        int r2 = (stand[i] >> 16) & 0xff; //高8位
        int g2 = (stand[i] >> 8) & 0xff; //中间8位
        int b2 = (stand[i]) & 0xff; //低8位

        if( (r2 >= r1) && (g2 >= g1) && (b2 >= b1) )
            count++;
    }

    return (double)count/detect.length; //计算符合条件的像素比例
}
```

去掉水印就是减去色彩饱和度差值。

## 9.2.5 数据导入 ECShop

网上商城应用托管，就是装了 EcShop，还有 MySQL 的 Linux 空间。卖给一些小的电子商务卖家，然后负责商城应用代码的升级和采集开发等。在自己的地盘上，采集开发的事情就好

商量了，卖给客户空间，然后对客户的商城进行维护。

要突出搜索相关专业性强的卖点，比如客户选择了你的服务，产品不用他操心，客户唯一需要做的就是自己网站的推广了。可以做采集，还可以做商品推荐挖掘。要有平台才能在上面加推应用。

ECShop 中的相关表包括：商品类别表（ecs\_category）、商品表（ecs\_goods）、商品品牌表（ecs\_brand）。

商品类别表 ecs\_category 说明如表 9-1 所示。

表 9-1 ecs\_category 说明

字段名	说明	值
cat_id	类别ID	
cat_name	类别名	
parent_id	父类别ID	
sort_order	排序方式	1
is_show	是否显示	1
grade	级别	0

商品表 ecs\_goods 说明如表 9-2 所示。

表 9-2 ecs\_goods 说明

字段名	名称	说明
cat_id	类别ID	
goods_name	商品名称	
click_count	单击次数	
goods_number	库存数量	
market_price	市场价	
shop_price	售价	
goods_brief	简介	
goods_desc	描述	可能包含div类似的复杂HTML格式
goods_thumb	缩略图	
goods_img	图	
original_img	原图	
is_real	是否真实商品	



续表

字段名	名称	说明
is_on_sale	是否上架	
is_alone_sale	是否单独销售	
add_time	添加时间	
is_delete	是否已经删除	
last_update	更新日期	
goods_type	商品类型	
seller_note	商品备注	

如果要想商品正常显示,则需要把 `is_delete` 字段的值设成 0。对于商品描述,需要修改 CSS 显示样式。

登录到 MySQL。

```
#./mysql -h118.145.6.205 -uroot -pPASSWORD
```

把数据从 Access 数据库找出来,然后插入到 MySQL 数据库。连接 MySQL 数据库的方法如下所示。

```
//设置连接数据库的用户名和密码
Properties props = new Properties();
props.put("user", "root"); //用户名
props.put("password", "SCL+rgve"); //密码

//连接参数,指定网址和数据库名
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://118.145.6.205/s2266", props); //IP 地址/数据库名
```

用 Commons VFS 把下载的商品图片上传到商城服务器。Commons VFS 底层使用 Apache Commons Net 实现 FTP 上传的功能,也可以直接使用 Apache Commons Net 把文件上传到 FTP。

登录到 FTP 服务器。

```
FTPClient ftp = new FTPClient();
ftp.connect("192.168.14.117"); //连接 FTP 服务器
ftp.login("admin", "123"); //登录
```

设置上传文件的类型是二进制。

```
ftp.setFileType(FTP.BINARY_FILE_TYPE);
```

默认上传到根路径,如果想要上传到其他路径,就改变当前工作路径。改变工作目录到图片所在目录如下所示。

```
ftp.changeWorkingDirectory("/admin/pic");
```

检查 FTP 连接是否成功。

```
int reply = ftp.getReplyCode();
if(FTPReply.isPositiveCompletion(reply)){
    System.out.println("连接成功");
}
```

上传文件到 FTP 服务器。

```
File f1 = new File(location); //本地文件
in = new FileInputStream(f1);
ftp.storeFile("test.jpg",in); //上传后的名字是 test.jpg
```

断开和服务器建立的连接。

```
ftp.logout();
```

## 9.2.6 采集淘宝

抓取淘宝中的热门商品,使用 HttpClient 得到网页内容,然后用 Jsoup 解析。

```
String url = "http://top1.search.taobao.com/?spm=0.0.0.0.4dtect";

String content = getContent(url); //使用 HttpClient 得到网页内容

Document doc = Jsoup.parse(content);
Elements links = doc.select("a[href]");

for (Element link : links) { //遍历每个链接
    String linkHref = link.attr("href"); //得到 href 属性中的值,也就是 URL 地址
    if(linkHref.startsWith("http://top.taobao.com/")){
        System.out.println(linkHref); //输出 URL 地址和锚点上的文字说明
    }
}
```

## 9.3 自动化行业采集

把信息抓到不同的类别下，举个例子。

- 新闻：资讯、行业动态等。
- 产品：产品库、新品。
- 下载：下载资料、提供下载的。
- 供求：供应的产品与需要求购的产品、买卖场。
- 文库：技术文章、解决方案、应用案例、专业论文、文摘、前沿技术。
- 企业：公司库、供应商。

例如，新闻按目录抓取 [http://mma.vogel.com.cn/news\\_list.html?py=52](http://mma.vogel.com.cn/news_list.html?py=52)。

企业 URL 有相似的特征，例如，<http://bjklha.corp.vogel.com.cn/corpinfo.html> 和 <http://igsl.corp.vogel.com.cn/corpinfo.html>。

## 9.4 社会化信息采集

采集豆瓣上的话题，提取其中用户之间的关系，把相关结果存入 SQLite 数据库。

## 9.5 微博爬虫

微博中有两类有用的信息：微博用户发布的信息和关注/粉丝信息，设计如下的存储接口。

```
public interface Target {  
    public void insertWeibo(User user,String content); //用户及发布的信息  
    public void insertRelation(User follow,User star); //追星族关注明星  
}
```

需要先登录才能抓取内容。整体流程如下所示。



```

HtmlUnitDriver driver = WeiboHelper.getDriver(); //得到登录后的下载器
String name = "周鸿祎";
String userURL = WeiboHelper.getUserUrl(driver, name); //根据名字得到用户首页
User user = new User(userURL);

Target target = new TargetDB(); //存入数据库
Extractor extractor = new Extractor(driver);
extractor.extractor(user, target); //提取微博信息

```

使用 Selenium 实现的登录代码。

```

public static HtmlUnitDriver getDriver(){
    String username = "yyyy@126.com";
    String password = "xxxxx";
    HtmlUnitDriver driver = new HtmlUnitDriver();
    driver.setJavascriptEnabled(true);
    driver.get("http://login.weibo.cn/login/");

    WebElement mobile = driver.findElementByCssSelector("input[name=mobile]");
    mobile.sendKeys(new CharSequence[] { username });
    WebElement pass = driver.findElementByCssSelector("input[name^=password]");
    pass.sendKeys(new CharSequence[] { password });
    WebElement rem = driver.findElementByCssSelector("input[name=remember]");
    rem.click();
    WebElement submit = driver.findElementByCssSelector("input[name=submit]");
    submit.click();
    return driver;
}

```

提取微博用户发布的信息，自动发现微博用户。

```

public class Extractor {
    public void extractor(User user, Target target){
        extractWeiBo(user, target); //提取发布内容
        extractRelation(user, target); //提取用户关系
    }
}

```

## 9.6 微信爬虫

通过 <http://weixin.sogou.com/weixin?type=1&query=xianbeitravel> 这样的入口抓微信公众号、朋友圈就抓到了。

## 9.7 海关数据

英国税务海关总署 (HMRC) 贸易统计局, 登录 <https://www.uktradeinfo.com> 网站可以查询到部分提单数据, 我国的国家商务部也存在部分海关数据。但这样的报关数据只有公司名称, 没有电子邮件或者网址。

验证抓取到的邮件地址的方法有 3 种。

- (1) 用正则表达式检查 Email 的格式。
- (2) 检查邮箱域名的有效性。通过 DNS 查找如 163.com 这个域名的 MX 记录。
- (3) 发送测试邮件。

用正则表达式检查 Email 的格式。\\w 与任何单词字符匹配, 包括下画线。用如下的正则表达式可以匹配上 luogang@gmail.com 文本。可以在 <http://regexpal.com/> 上在线测试这个正则表达式。

```
\\w+@[\\w+\\.\\w+]+
```

还需要 “.” 和 “-” 两个字符。例如, 邮箱 zhangshna.Mr@163.com, 在 @ 符号之前还有个点 “.”。

检查 Email 的格式的语句如下所示。

```
String mailTo = "abc@sina.com.cn";  
System.out.println(mailTo.matches( "[\\w[.-]]+@[\\w[.-]]+\\.([\\w]+)" )); //输出 true
```

查找域名的 MX 记录。

```

Lookup lookup = new Lookup(hostName, Type.MX);
lookup.run();
if (lookup.getResult() != Lookup.SUCCESSFUL) {
    return false;
}

```

发送测试邮件。

```

Socket socket = new Socket(host, 25);
BufferedWriter out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));
BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
in.readLine();
out.write("HELO Sender ");
out.flush();
in.readLine();
out.write("MAIL FROM:<wanderman1836@yahoo.com> ");
out.flush();
in.readLine();
out.write("RCPT TO:<" + mailTo + "> ");
out.flush();
String r = in.readLine();
out.write("QUIT ");
out.flush();

out.close();
in.close();
socket.close();
if (!r.startsWith("250")) {
    return false;
}
else {
    return true;
}

```

## 9.8 医药数据

PubMed 是美国国家医学图书馆 (NLM) 下属的国家生物技术信息中心 (NCBI) 开发的医药



论文数据库。这个数据库提供了编程接口返回 XML 格式的论文描述信息。

可以通过 <http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi> 接口按查询关键词抓取论文信息。

首先取得结果数量。

```
public static int getResultNum(CloseableHttpClient client, String term)
    throws Exception {
    String data = new URI(null, term, null).toASCIIString();
    String url = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?term="
        + data + "&retmax=10&retstart=1";

    String content = HttpUtil.getContent(client, url);

    Document doc = Jsoup.parse(content);
    Element e = doc.getElementsByTag("Count").first();
    String countStr = e.text();

    return Integer.parseInt(countStr);
}
```

使用默认的 `HttpClient` 对象会报 `Invalid cookie header` 的警告。解决 `Cookie rejected` 警告最简单的办法是，设置个空 `Cookie`，参考代码如下所示。

```
//采用用户自定义Cookie策略，只是使 cookie rejected 的报错不出现
CookieSpecProvider easySpecProvider = new CookieSpecProvider() {
    @Override
    public CookieSpec create(org.apache.http.protocol.HttpContext arg0) {
        return null;
    }
};

Registry<CookieSpecProvider> registry =
    RegistryBuilder.<CookieSpecProvider>create()
        .register("easy", easySpecProvider)
        .build();

CloseableHttpClient client = HttpClients.custom()
    .setDefaultCookieSpecRegistry(registry)
    .build();
```

然后根据结果数量设定要抓取多少条结果。

```
public static void insertIDS(CloseableHttpClient client, String term) throws Exception{
    Connection con = DBUtil.getConnect();
    int resultNum = getResultNum(client, term);

    String data = new URI(null, term, null).toASCIIString();
    String url = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?term="
        + data + "&retmax="+resultNum+"&retstart=1";
    String content = HttpUtil.getContent(client, url);
    Document doc = Jsoup.parse(content);
    Elements tables = doc.getElementsByTag("Id");

    String sql = "insert into paper(id)values(?)";
    PreparedStatement insertStmt = con.prepareStatement(sql);

    for (Element e : tables) {
        String id = e.text();
        insertStmt.setString(1, id);
        insertStmt.executeUpdate();
    }
    con.commit();
}
```

## 9.9 本章小结

券商的 Web 行情交易系统可以自动登录, 然后实现自动交易。Selenium 实现浏览器行为自动化。

科恩·萨瑟兰线段裁剪算法是计算机图形学直线段裁剪算法的一种, 由丹·科恩 (Dan Cohen) 和伊凡·苏泽兰 (Ivan Sutherland) 两人提出。

## 后 记

作者最早使用网络爬虫是在 2000 年的时候，帮助华大基因公司抓取日本水稻基因数据。当时，使用了一个叫作 Teleport Pro 的离线浏览器软件，这个软件的功能类似 Crawler4J，不过 Teleport Pro 用图形化能更直观地显示抓取进度。

很多公司的网络爬虫相关的开发岗位并不长久，所以鼓励技术人员自己攒钱来一起创业。少买一辆车，创业的钱就出来了，但不要单打独斗。很多成功的大公司都不是一个人独自做起来的。

代码和技术的完善是一个一直持续并且在很长时间内不会终结的过程，希望我们能一路相伴。



## 博文视点诚邀精锐作者加盟

以书为证彰显卓越品质

《C++Primer (中文版) (第5版)》、《淘宝技术这十年》、《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、管辉Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与IT业的蓬勃发展紧密相连。

十年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金之计算机图书的风向标杆:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟,与大师并列于IT专业出版之颠。

### 英雄帖

江湖风云起,代有才人出。

IT界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享IT心得

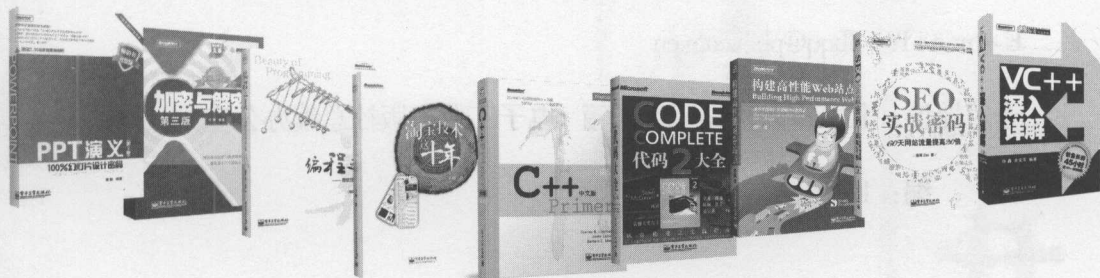
### 专业的作者服务

博文视点自成立以来一直专注于IT专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照IT技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

**善待作者**——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

**尊重作者**——我们尊重每一位作者的技术实力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

**提升作者**——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



### 联系我们

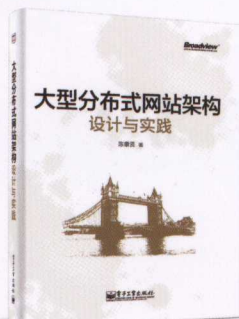
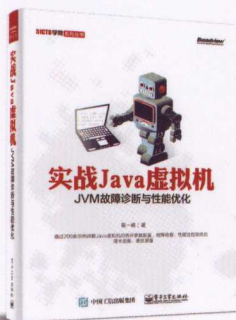
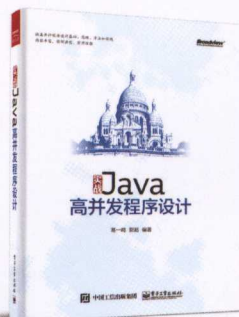
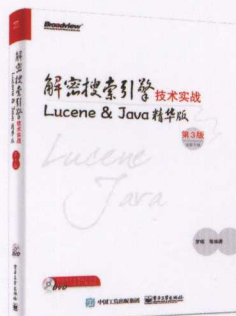
博文视点官网: <http://www.broadview.com.cn>

CSDN官方博客: <http://blog.csdn.net/broadview2006/>

投稿电话: 010-51260888 88254368

投稿邮箱: [jsj@phei.com.cn](mailto:jsj@phei.com.cn)





欢迎反馈意见或投稿  
邮箱: [dongying@phei.com.cn](mailto:dongying@phei.com.cn)  
电话: 010-88254047  
微信号: yingzidd

# 网络爬虫全解析

## 技术、原理与实践



本书介绍了如何开发网络爬虫。内容主要包括开发网络爬虫所需要的Java语法基础和网络爬虫的工作原理，如何使用开源组件HttpClient和爬虫框架Crawler4j抓取网页信息，以及针对抓取到的文本进行有效信息的提取。为了扩展抓取能力，本书介绍了实现分布式网络爬虫的关键技术。

另外，本书介绍了从图像和语音等多媒体格式文件中提取文本信息，以及如何使用大数据技术存储抓取到的信息。最后，以实战为例，介绍了如何抓取微信和微博，以及在电商、医药、金融等领域的案例应用。其中，电商领域的应用介绍了使用网络爬虫抓取商品信息入库到网上商店的数据库表。医药领域的案例介绍了抓取PubMed医药论文库。金融领域的案例介绍了抓取股票信息，以及从年报PDF文档中提取表格等。

本书适用于对开发信息采集软件感兴趣的自学者，也可以供有Java或程序设计基础的开发人员参考。



博文视点Broadview



@博文视点Broadview



策划编辑：董 英

责任编辑：徐津平

封面设计：李 玲

上架建议：网络爬虫

ISBN 978-7-121-31071-3



9 787121 310713 >

定价：79.00元